MARVIN FROMMHOLD

MODELLIERUNG VON TOURISTISCHEN MERKMALEN IN RDF UND EVALUATION AM ANWENDUNGSFALL VAKANTIELAND

MASTERARBEIT
Fakultät für Mathematik und Informatik
Institut für Informatik

Marvin Frommhold: *Modellierung von touristischen Merkmalen in RDF und Evaluation am Anwendungsfall Vakantieland,* Masterarbeit, 8. November 2010.

E-MAIL:
marvin.frommhold@gmail.com

BETREUER:
Michael Martin
Dr. Sören Auer
Prof. Dr. Klaus-Peter Fähnrich

ZUSAMMENFASSUNG

Die Tourismus-Domäne ist eine sehr Informations-intensive Industrie. Im eTourismus-Bereich ist daher eine mächtiges Datenschema zur geeigneten Speicherung und Abfrage der Daten erforderlich. Bis vor einigen Jahren haben dafür relationale Datenbanken und Dokumenten-zentrierte Systeme ausgereicht. Für einen Touristen spielt aber heute das schnelle und einfache Befriedigen seines Informationsbedürfnisses eine immer größer werdende Rolle. Aus diesem Grund wird mehr und mehr auf den Einsatz von semantischen Technologien im Bereich des eTourismus gesetzt. So geschehen auch bei der Transformation des Tourismus-Portals *vakantieland.nl* zu einer semantischen Web-Applikation. Eine solche Umstellung bringt jedoch auch neue Probleme mit sich. Zum Beispiel die Frage, wie touristische Informationen geeignet mit Hilfe des Resource Description Frameworks (RDF) modelliert werden können. In dieser Arbeit wird dieser Frage in Bezug auf die Modellierung von Eigenschaften von touristischen Zielen nachgegangen. Dazu wird eine bestehende eTourismus-Ontologie analysiert und basierend darauf ein geeignetes Schema definiert. Anschließend wird die Ontologie einer Evolution unterzogen, um diese an das neue Schema anzupassen. Um den Nutzen des Tourismus-Portals zusätzlich zu erhöhen, werden außerdem die bereits existierenden Filterfunktionen erweitert.

STICHWORTE: Semantic Web, RDF, eTourismus, Ontologie-Evolution, Vakantieland

INHALTSVERZEICHNIS

1 EINLEITUNG

Inhaltsangabe

Applikationen, welche mannigfaltige Informationsstrukturen verarbeiten, benötigen ein flexibles Datenschema. Semantische Web-Applikationen, welche derartige Informationsstrukturen verarbeiten, verwenden in fast allen Fällen das Resource Description Framework (RDF) zur Modellierung der Informationen. Es ist dabei aber nicht immer sofort ersichtlich beziehungsweise absehbar, inwieweit die aktuell gewählte Modellierung der Informationen in Zukunft noch den gestellten Anforderungen genügt. Tritt diese Situation ein, so müssen das Schema und die darauf aufbauenden Daten den neuen Anforderungen angepasst werden. In dieser Arbeit soll ein solcher Fall vorgestellt und eine Anpassung eines bereits existierenden Schemas durchgeführt werden. Dazu werden nun zunächst die Motivation und Ziele vorgestellt, bevor im Anschluss kurz auf den Aufbau der einzelnen Kapitel eingegangen wird.

1.1 MOTIVATION

Ein Beispiel für eine semantische Web-Applikation ist das niederländische Tourismusportal *vakantieland.nl*, dass eine eigens dafür geschaffene eTourismus-Ontologie verwendet. Zu den in RDF gespeicherten Informationen über touristische Ziele gehören unter anderem zwei Taxonomien. Eine Taxonomie bildet dabei eine Klassenhierarchie der Typen von touristischen Zielen ab, die andere eine Hierarchie von Eigenschaften, auch touristische Merkmale genannt, zu diesen Zielen. Da der Datenbestand aus über 20.000 touristischen Zielen besteht, sind effektive und ohne Vorkenntnis nutzbare Filterfunktionen nötig, um dem Benutzer die Möglichkeit zu bieten, aus der Fülle die für ihn relevanten Informationen herauszusuchen. Für die Typenhierarchie existiert bereits eine geeignetee Filterfunktion. Jedoch ist aktuell noch keine Funktionalität vorhanden, auch nach den Eigenschaften der touristischen Ziele zu filtern. Hier liegt es auf der Hand, einen solchen Filter auch für die touristischen Merkmale umzusetzen. Die derzeit vorliegende Taxonomie für diese Merkmale ist aber aus Sicht der Ontologie-Entwicklung sowie aus inhaltlichen Aspekten fehler-

haft und weist Inkonsistenzen auf. In dieser Arbeit sollen diese Probleme aufgedeckt und durch eine Anpassung der Ontologie, konkret das Schema für die Merkmale, behoben werden.

1.2 ZIELE

Das Ziel dieser Arbeit ist die Entwicklung eines neuen Schemas zur Modellierung von touristischen Merkmalen mit Hilfe von RDF. Als konkreter Anwendungsfall dient dabei die eTourismus-Ontologie der semantischen Web-Applikation *vakantieland.nl*. Mit Hilfe von Ontologie-Entwurfsmustern sollen Vorgaben zur Modellierung der unterschiedlichen Gruppen von Merkmalen definiert werden. Durch das sich daraus zusammensetzende und klar definierte Schema soll die Qualität und Wartbarkeit der Merkmalsdaten gesteigert werden. Die Anpassung der Daten der Ontologie soll dabei durch eine Evolution geschehen, für die ein geeignetes Programm zur Durchführung der Anpassungen entwickelt werden muss. Im Anschluss an diese Evolution soll dann eine Filterfunktion für die touristischen Merkmale in die Applikation *vakantieland.nl* integriert werden. Diese Implementierung dient dabei gleichzeitig als Evaluation, um zu zeigen, dass aus Sicht des Ontologie-Designs eine generische und fehlerfreie Modellierung der Merkmalsangaben vorliegt.

1.3 GLIEDERUNG DER ARBEIT

In Kapitel 2 werden zunächst grundlegende Begriffe und Technologien, die für das Verständnis dieser Arbeit von Bedeutung sind, vorgestellt. Der bereits angesprochene Anwendungsfall *vakantieland.nl* und die dazu gehörige Ontologie werden in Kapitel 3 näher betrachtet. In Kapitel 4 wird der aktuelle Stand zur Evolution von Ontologien besprochen. Außerdem werden einige Werkzeuge, die eine solche Evolution unterstützen, vorgestellt, von denen wiederum eines für einen Einsatz in dieser Arbeit ausgewählt und genauer erläutert wird. Die Anforderungen für das neu zu entwickelnde Schema, das Programm zur Evolution und der zu implementierenden Filterfunktion für die Merkmale werden in Kapitel 5 erläutert. Daraufhin werden basierend auf einer Analyse des alten Schemas in Kapitel 6 verschiedene Ontologie-Entwurfsmuster zur Modellierung der Merkmale definiert. Entsprechend dieser Entwurfsmuster werden in Kapitel 7 Algorithmen zur Anpassung der Merkmale an das neue Schema erarbeitet. Die Umsetzung dieser Algorithmen in einem Programm zur Evolution und weiterer Implementierungsarbeiten werden in Kapitel 8 dargelegt. Abschließend werden in Kapitel 9 zunächst die Ergebnisse dieser Arbeit zusammengefasst und einer kurzen Evaluation unterzogen, bevor ein kleiner Ausblick auf mögliche zukünftige Entwicklungen gegeben wird.

2 GRUNDLAGEN

Inhaltsangabe

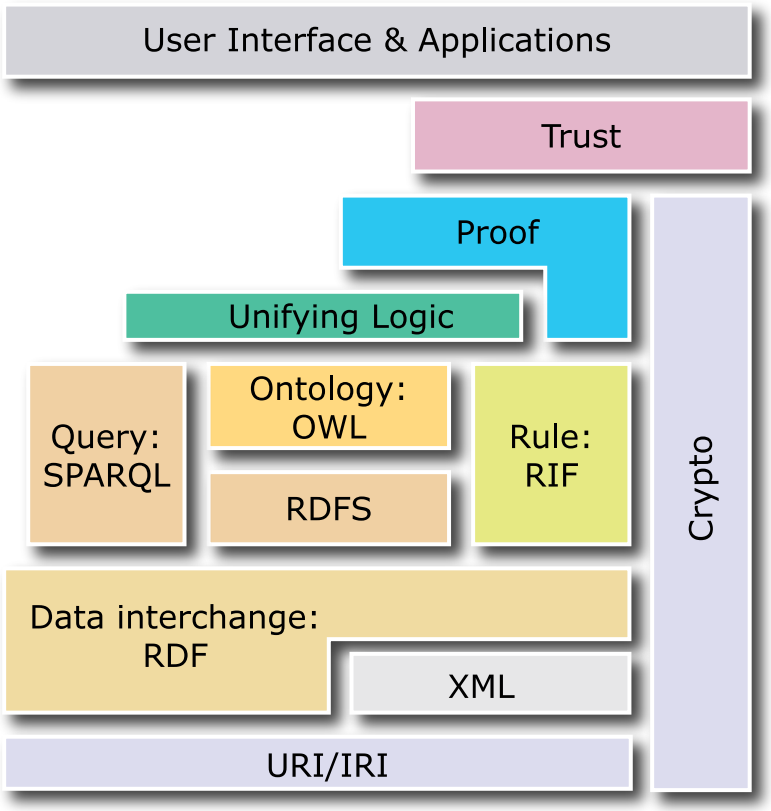
Das folgende Kapitel stellt die grundlegenden Begriffe vor, welche für das Verständnis der übrigen Kapitel notwendig sind. Zu Beginn wird das Semantic Web und dessen Vision erläutert. Anschließend werden die Konzepte sowie Technologien vorgestellt, die zur Verwirklichung dieser Vision entwickelt wurden.

2.1 SEMANTIC WEB

Der Begriff Semantic Web wurde zu Beginn des neuen Jahrtausends von Tim Berners-Lee, dem Begründer des World Wide Web, beschrieben [Berners-Lee u. a., 2001]. Es bezeichnet dabei kein eigenständiges Web, sondern stellt eine Erweiterung des bisherigen World Wide Web, kurz WWW, dar, welches als ein weit verzweigtes Netz aus einer unüberschaubaren Menge von untereinander verlinkten Dokumenten angesehen werden kann. Da diese Dokumente zum größten Teil aus freiem Text bestehen, welche zwar von Menschen, aber nicht bzw. nur schwer von Computern interpretiert werden können, entstand die Vision eines „maschinenlesbaren“ Webs [Berners-Lee u. Fischetti, 1999]:

> I have a dream for the Web (in which computers) become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.
>
> – Tim Berners-Lee, 1999

Hinter dieser Vision steckt also die Idee, das Netz aus Dokumenten zu erweitern, indem es mit zusätzlichen semantischen Informationen ausgestattet wird. Dadurch soll es Computern ermöglicht werden, die zur Verfügung stehenden Daten zu interpretieren. Des Weiteren wird dadurch eine riesige verteilte Datenbasis geschaffen, in der alle Daten miteinander kombiniert und abgefragt werden können. Diese vernetzte Datenbasis wird auch als Linked Data [Berners-Lee, 2006] bezeichnet. Um die Vision des Semantic Webs zu verwirklichen, wurden verschiedene, teilweise aufeinander aufbauende, Konzepte und Technologien entwickelt. Die Abbildung 1 zeigt das Semantic Web-Schichtenmodell [Berners-Lee, 2000], welches stetig weiterentwickelt wird. Im Folgenden werden die für diese Arbeit relevanten Schichten näher beschrieben.



Quelle: `http://www.w3.org/2007/03/layerCake.svg`

Abbildung 1: Das Semantic Web-Schichtenmodell.

2.2 UNIFIED RESOURCE IDENTIFIER

Unified Resource Identifier, kurz URI, steht für einen eindeutigen Bezeichner für eine Ressource, diese kann sowohl physisch als auch abstrakt sein. URIs besitzen einen generischen Standard [Berners-Lee u. a., 2005] und

dienen zur Identifikation von Ressourcen über Internet-Dienste. Der allgemeine Syntax einer URI besteht aus bis zu vier Teilen:

```
1 <scheme name>:<hierarchical part>[?<query>][#<fragment>]
```

Wie der Syntax zu entnehmen ist, sind die beiden ersten Teile, der Schema- und Hierarchie-Teil, pflicht. Diese werden durch einen Doppelpunkt getrennt. Der Anfrage- (query) und Fragment-Teil sind dagegen optional. Der Schema-Teil gibt den Typ der URI an, welcher wiederum die Interpretation des Teils hinter dem Doppelpunkt festlegt. Bekannte Schemata sind `http` für Internet-Adressen, `mailto` für Email-Adressen oder `ftp` für Adressen zur Übertragung von Dateien.
Im Semantic Web werden URIs dazu verwendet, Ressourcen weltweit eindeutig zu bezeichnen.

2.3 RESOURCE DESCRIPTION FRAMEWORK

Das Resource Description Framework (RDF) bedeutet sinngemäß übersetzt: „System zur Beschreibung von Ressourcen“. Es bezeichnet eine Menge von Standards [Klyne u. Carroll, 2004; Hayes u. McBride, 2004; Manola u. Miller, 2004; Brickley u. Guha, 2004; Beckett, 2004; Grant u. Beckett, 2004], welche durch das World Wide Web Consortium (W3C) im Jahre 2004 verabschiedet wurden. RDF erweitert die Linkstruktur des Webs durch Verwendung von URIs zur Bezeichnung von Beziehungen zwischen zwei Ressourcen sowie der Ressourcen selbst. Diese Verbindung zweier Ressourcen durch eine Beziehung, auch Relation genannt, wird als Tripel bezeichnet. Abbildung 2 zeigt eine grafische Darstellung eines Tripels.

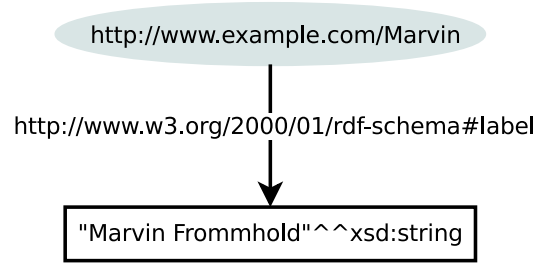


Abbildung 2: Beispiel für ein einfaches Tripel mit einem Literal als Objekt.

Die drei Komponenten eines solchen RDF-Tripels sind:

SUBJEKT (engl. subject): Die Ressource, über die eine Aussage getroffen wird. Erlaubt sind an dieser Stelle URIs oder anonyme Knoten.

PRÄDIKAT (engl. predicate oder auch property): Das Prädikat bezeichnet eine Beziehung von einer Ressource zu einer anderen und ist damit immer gerichtet. Es können nur URIs an der Prädikat-Position verwendet werden.

OBJEKT (engl. object): Der Wert der Beziehung. Es kann sich hierbei wiederum um eine Ressource handeln (URIs oder anonyme Knoten) oder um ein Literal (Zeichenkette).

Unter einem anonymen Knoten (engl. blank node) versteht man weder eine URI noch ein Literal, es handelt sich dabei um einen eindeutigen Identifikator innerhalb eines Graphen. Meist wird eine konsistente Menge von Prädikaten, welche zusammen ein komplexes Datum repräsentieren, über einen solchen anonymen Knoten an eine Ressource gebunden. Jedoch ist der Einsatz von anonymen Knoten problematisch [Arenas u. a., 2010]. So stimmt in manchen Situationen die Semantik von SPARQL[1] nicht mit der empfohlenen Semantik des W3C Consortiums für anonyme Knoten überein.

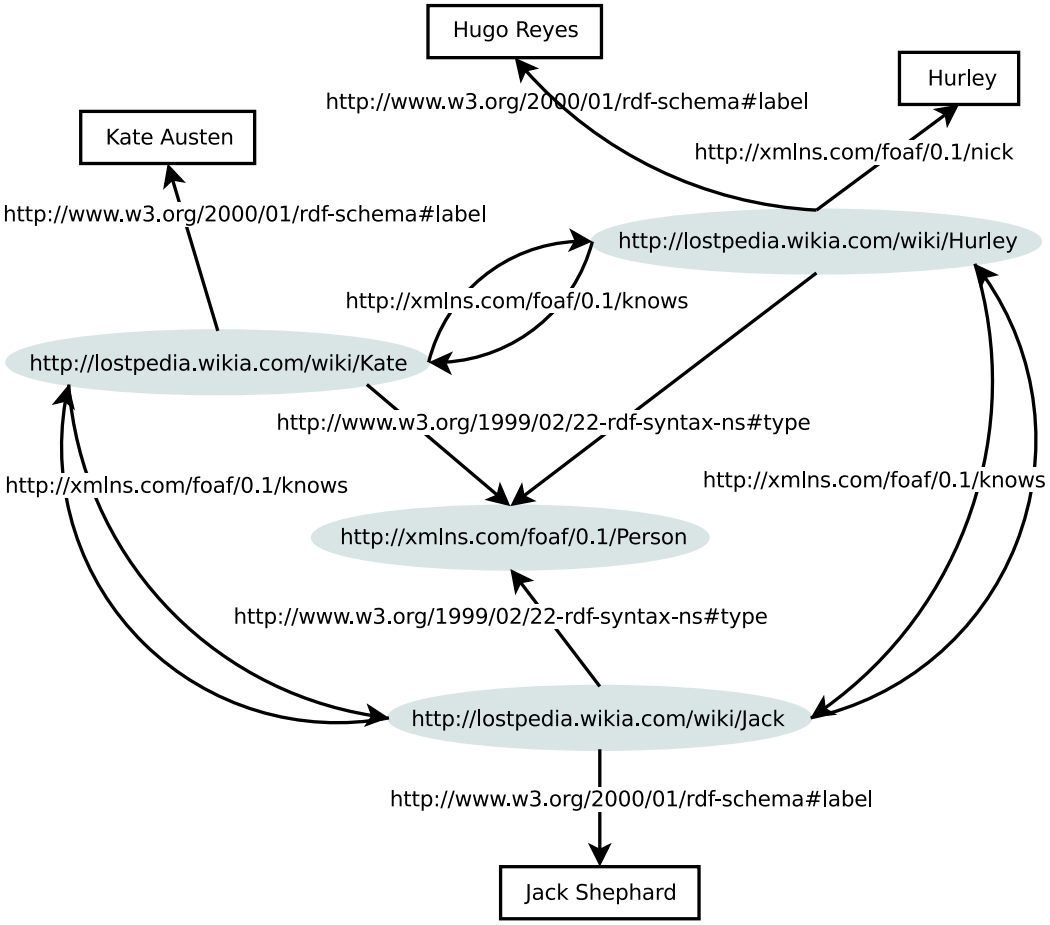


Abbildung 3: Beispiel für einfachen RDF-Graphen.

Einen weiteren Bestandteil von RDF stellen die Literale dar. Sie werden dazu verwendet um Zahlen, Datumsangaben und Zeichenketten zu repräsentieren. Literale können getypt oder ungetypt sein. Ein ungetyptes Literal stellt eine Zeichenkette mit optionaler Sprachangabe dar, sie werden für Klartext verwendet. Getypte Literale sind ebenfalls Zeichenketten,

1 siehe Paragraph SPARQL Anfragesprache im Abschnitt 2.6

welche aber, wie der Name bereits ausdrückt, durch einen Typ gekennzeichnet sind. Der (Daten-)Typ gibt dabei an, wie die Zeichenkette jeweils zu interpretieren ist.
Die verlinkte Struktur zwischen mehreren Tripel ergibt einen gerichteten, markierten Graphen. Die Abbildung 3 zeigt einen solchen einfachen Beispiel-Graphen. Die Kanten repräsentieren dabei die Prädikate und die Knoten die Subjekte bzw. Objekte aus den Tripeln. Ein solcher visueller Graph stellt die einfachste Form der Serialisierung eines RDF-Modells dar. So gibt es noch weitere Serialisierungsformen wie zum Beispiel RDF/XML [Beckett, 2004], Notation 3 [Berners-Lee u. Connolly, 2008] oder den Turtle-Syntax [Beckett u. Berners-Lee, 2008]. Das Listing 1 zeigt den Graphen aus der Abbildung 3 in Turtle-Syntax. Auf Grund der guten Lesbarkeit wird der Turtle-Syntax, insofern nicht anders angegeben, für alle Listings, welche RDF darstellen, eingesetzt.

```
1  @prefix lost: <http://lostpedia.wikia.com/wiki/> .
2  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
5
6  lost:Kate    rdfs:label "Katherine Anne Austen" ;
7               rdf:type foaf:Person ;
8               foaf:nick "Kate";
9               foaf:knows ex:Jack ;
10              foaf:knows ex:Hurley .
11
12 lost:Jack    rdfs:label "Dr. Jack Shephard" ;
13              rdf:type foaf:Person ;
14              foaf:knows ex:Kate ;
15              foaf:knows ex:Hurley .
16
17 lost:Hurley  rdfs:label "Hugo Reyes" ;
18              rdf:type foaf:Person ;
19              foaf:nick "Hurley";
20              foaf:knows ex:Kate ;
21              foaf:knows ex:Jack .
```

Listing 1: Beispiel eines RDF-Modells in Turtle-Syntax.

2.4 RDF-SCHEMA

Da RDF keine Mechanismen zur Beschreibung von Relationen und den Beziehungen zwischen diesen Relationen und anderen Ressourcen bereitstellt, wurde RDF-Schema (RDFS) [Brickley u. Guha, 2004] als semantische Erweiterung zu RDF entwickelt. RDFS dient zur Beschreibung von RDF Vokabularen, wobei es selber auch ein Vokabular darstellt. Es bietet ein Klassen- und Prädikatsystem, welches ähnlich zu denen von Objekt-

Klasse	Beschreibung
`rdfs:Resource`	Klasse aller RDF-Ressourcen. Instanz der Klasse `rdfs:Class`.
`rdfs:Class`	Klasse aller Klassen. Instanz von sich selbst.
`rdfs:Literal`	Klasse aller Literale. Literale können getypt oder ungetypt sein. Ein getyptes Literal ist Instanz einer Datentyp-Klasse. Instanz der Klasse `rdfs:Class`, Unterklasse von `rdfs:Resource`.
`rdfs:Datatype`	Klasse aller Datentypen. Ist zugleich Instanz als auch Unterklasse von `rdfs:Class`. Jede Instanz dieser Klasse ist auch Unterklasse von `rdfs:Literal`.
`rdf:XMLLiteral`	Klasse aller XML-Literale [Bray u. a., 2008]. Instanz von `rdfs:Datatype` und Unterklasse von `rdfs:Literal`.
`rdf:Property`	Klasse aller Prädikate. Instanz von `rdfs:Class`.

Tabelle 1: Definierte Klassen des RDF-Schema Vokabulars.

orientierten Programmiersprachen wie zum Beispiel Java[2] ist. Jedoch unterscheidet es sich zu diesen dahingehend, dass es nicht Klassen in Bezug auf die Eigenschaften seiner Instanzen definiert, sondern Relationen beschreibt und den Bezug zu Klassen von Ressourcen. Dieser eher Relationen-zentrierte Ansatz erlaubt es, neue Eigenschaften für bereits existierende Ressourcen zu definieren, ohne die ursprüngliche Beschreibung einer Ressource verändern zu müssen.
Ressourcen lassen sich allgemein in Gruppen einteilen, sogenannte Klassen. Mitglieder einer solchen Klasse werden als Instanz der Klasse bezeichnet. Bei Klassen handelt es sich ebenfalls um Ressourcen und sie können Instanz von sich selbst sein. Die Tabelle 1 zeigt alle Klassen, welche durch das RDFS Vokabular definiert sind. Alle vordefinierten Prädikate des RDFS Vokabulars sind in der Tabelle 2 angegeben.

2.5 WEB ONTOLOGY LANGUAGE

Um die Vision des Semantic Webs zu verwirklichen, wird eine mächtige und ausdrucksstarke Beschreibungssprache benötigt. Allerdings reicht dafür das RDF-Schema Vokabular nicht aus. So können keine weitreichenderen Einschränkungen für Prädikate über den Werte- und Definitionsbereich hinaus festgelegt werden, beispielsweise Kardinalitätsrestriktionen. Die Definition von komplexen Klassen ist ebenfalls nicht möglich. Aus diesem Grund wurde die Web Ontology Language (OWL) [Motik u. a., 2008] entwickelt. Mit Hilfe dieser Ontologiesprache kann die Semantik

2 http://www.java.com/

Prädikat	Beschreibung
`rdfs:range`	Legt den Wertebereich für ein Prädikat fest. Der Wertebereich von `rdfs:range` ist `rdfs:Class`, der Definitionsbereich ist `rdf:Property`.
`rdfs:domain`	Legt den Definitionsbereich für ein Prädikat fest. Der Werte- und Definitionsbereich entspricht dem von `rdfs:range`.
`rdf:type`	Besagt, dass eine Ressource Instanz einer bestimmten Klasse ist. Der Wertebereich ist `rdfs:Class`, der Definitionsbereich ist `rdfs:Resource`.
`rdfs:subClassOf`	Wird zur Erzeugung von Klassenhierarchien verwendet. Alle Instanzen einer Unterklasse sind auch Instanzen der Superklasse. Der Werte- und Definitionsbereich ist auf `rdfs:Class` festgelegt.
`rdfs:subPropertyOf`	Wird zur Erzeugung von Prädikatshierarchien verwendet. Der Werte- und Definitionsbereich ist auf `rdf:Property` festgelegt.
`rdfs:label`	Dient zur Angabe eines Menschen-lesbaren Namens für eine Ressource. Der Wertebereich ist `rdfs:Literal`, der Definitionsbereich ist `rdfs:Resource`.
`rdfs:comment`	Dient zur Angabe einer Menschen-lesbaren Beschreibung für eine Ressource. Der Wertebereich ist `rdfs:Literal`, der Definitionsbereich ist `rdfs:Resource`.

Tabelle 2: Definierte Prädikate des RDF-Schema Vokabulars.

von Klassen, Prädikaten und kompletten Ontologien formal beschrieben werden und zwar in einem viel höheren Maße, als es mit RDF(S) möglich wäre. Eine Ontologie bezeichnet im Kontext des Semantic Webs eine explizite formale Spezifikation einer Konzeptualisierung [Gruber, 1993], stellt also eine Art Wissensbasis dar, welche in digitaler und formaler Form zwischen Anwendungen und Diensten ausgetaucht werden kann. Sie besitzt häufig auch Inferenz- und Integritätsregeln mit denen Schlussfolgerungen gezogen werden können. Um solche Ontologien zu erstellen, bietet OWL eine Reihe von Sprachelementen. Eine Vorstellung aller Elemente liegt an dieser Stelle außerhalb des Gegenstandsbereichs dieser Arbeit. Die Tabelle 3 bietet jedoch einen Auszug mit den Elementen, welche auch in dieser Arbeit zur Anwendung kommen.

Element	Beschreibung
`owl:Class`	Deklariert eine Ressource als OWL Klasse.
`owl:ObjectProperty`	Klasse von Prädikaten, welche Individuen mit Individuen verbinden.
`owl:DatatypeProperty`	Klasse von Prädikaten, welche Individuen mit Werten verbinden.
`owl:imports`	Prädikat, welches eine weitere OWL Ontologie referenziert. Die referenzierte Ontologie wird als Teil der importierenden Ontologie interpretiert.
...	...

Tabelle 3: Auszug aus den vordefinierten Elementen der Ontologiesprache OWL.

2.6 SPARQL

Der Name SPARQL ist ein rekursives Akronym und steht für SPARQL Protocol and RDF Query Language. Die SPARQL Spezifikation besteht aus drei Teilen, einer Anfragesprache [Prud'hommeaux u. Seaborne, 2008], einem XML-basierten Format [Beckett u. Broekstra, 2008] zum Binden von Ergebnissen der Anfragesprache und einem Protokoll [Clark u. a., 2008] zur Anfrage von SPARQL-Endpunkten.
Das Protokoll und das XML-Rückgabeformat haben für diese Arbeit keine Relevanz und werden deshalb nicht weiter erläutert. Auf die SPARQL-Anfragesprache soll aber an dieser noch einmal genauer eingegangen werden.
Die SPARQL-Anfragesprache kann dazu verwendet werden, um Anfragen an diverse Datenquellen zu senden. SPARQL besitzt die Möglichkeit zur Anfrage von benötigten und optionalen Graphmustern zusammen mit ihren Konjunktionen und Disjunktionen. Des Weiteren werden eine erweiterte Werte-Prüfung sowie die Einschränkung der Anfrage auf bestimmte RDF-Graphen unterstützt. Grundsätzlich besteht eine Anfrage aus einer Menge von Tripel-Mustern. Ein Tripel-Muster repräsentiert dabei ein RDF-Tripel aus Subjekt, Prädikat und Objekt, wobei diese jeweils durch Variablen ersetzt werden können. Variablen werden durch ein „?“ gefolgt von einem Namen gekennzeichnet. Die Tripel-Muster können nun durch geschweifte Klammern zu Graph-Mustern zusammengefügt werden. Diese einzelnen Graph-Muster können wiederum mit Hilfe verschiedener Schlüsselwörter (`UNION` und `OPTIONAL`) kombiniert werden. Zusätzlich gibt es eine Möglichkeit zur Filterung von Ergebnis-Tupeln (`FILTER`). Hier stehen verschiedene Funktionen zur Verfügung, mit denen die Variablen ausgewertet werden können. Im Listing 2 ist als Beispiel eine `SELECT`-Anfrage auf den Graphen aus der Abbildung 3 zu sehen. Die entsprechende Ergebnismenge zeigt die Tabelle 4.

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3
4  SELECT ?name ?nick
5  WHERE {
6   ?person a foaf:Person .
7   ?person rdfs:label ?name .
8   OPTIONAL {
9    ?person foaf:nick ?nick .
10  }
11 }
```

Listing 2: Beispiel einer SPARQL `SELECT`-Anfrage.

Mit `SELECT` können Anfragen entsprechend den angegeben Tripel-Mustern gestellt werden und die Ergebnismenge wird in einer Liste an die entsprechenden Variablen gebunden. Durch `CONSTRUCT` wird ein RDF-Graph gemäß einer vorgegeben Graph-Vorlage erzeugt. Mit Hilfe von `ASK` kann geprüft werden, ob ein Graph-Muster zu einer Ergebnismenge führt. Die letzte Anfrageart `DESCRIBE` erzeugt ein Ergebnis, welches RDF-Daten über Ressourcen enthält.

name	nick
Katherine Anne Austen	Kate
Dr. Jack Shephard	
Hugo Reyes	Hurley

Tabelle 4: Ergebnis der SELECT-Anfrage aus dem Listing 2.

3 BETRACHTUNGSGEGENSTAND

Inhaltsangabe

Im diesem Kapitel erfolgt die Vorstellung der als Anwendungsfall dienenden Applikation Vakantieland, sowie einiger für die Arbeit relevanter Begriffsdefinitionen. Im Anschluss werden die in dieser Applikation verwendeten und definierten eTourismus-Vokabulare erläutert. Zusammen mit den Instanzdaten bilden diese Vokabulare die eTourismus-Ontologie, welche durch diese Arbeit einer Evolution unterzogen werden soll.

3.1 VAKANTIELAND

Die Web-Applikation *vakantieland.nl* (kurz Vakantieland) ist ein in den Niederlanden entwickeltes und betriebenes eTourismus-Portal. Mit eTourismus wird im Allgemeinen der Einsatz von Informations- und Kommunikationstechnologien im touristischen Umfeld bezeichnet. Das Portal bietet Benutzern einen Überblick zu touristischen Zielen, sogenannten *Point of Interests*, auch kurz als POI bezeichnet. Zu jedem dieser POIs werden zusätzlich Informationen bereitgestellt. Dazu gehören der Name, Kontaktdaten, eventuell eine textuelle Beschreibung und Angaben zur geographischen Lage. Des Weiteren besitzen alle POIs eine Klassifizierung (Hotel, Museum, Campingplatz, ...) und Angaben zu deren touristischen Merkmalen. Als touristisches Merkmal, kurz Merkmal, wird in dieser Arbeit eine Eigenschaft eines POI bezeichnet, durch die er sich von anderen, ähnlichen POIs unterscheidet, aber auch gemeinsam haben kann. Solche Eigenschaften können Ausstattungsmerkmale wie ein Swimming-Pool, Sportangebote, Entfernungsangaben zum nächstgelegen Strand, Verbote, Dienstleistungen und vieles mehr darstellen. In der ursprüngliche Version von Vakantieland wurden alle Daten in einer relationalen Datenbank gehalten. Im Zuge der Arbeiten aus [Martin, 2007] wurde die komplette Applikation in PHP[1] neugeschrieben. Dabei wurde die alte Datenbasis mit den Informationen zu über 20.000 POIs semantisch annotiert und in einer Ontologie (siehe Abschnitt 3.2) gespeichert. Bei dieser neuen Implementierung handelt es sich um eine semantische Web-Applikation [Martin u. Auer,

eTourismus

touristisches Merkmal

1 http://www.php.net/

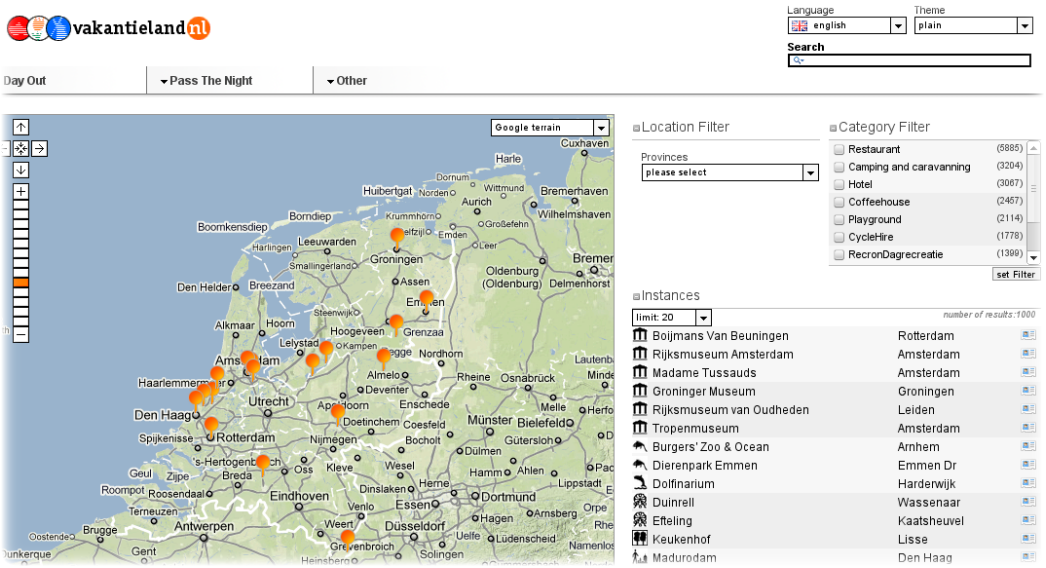


Abbildung 4: Die Startseite der semantischen Web-Applikation Vakantieland.

2010], welche seitdem stetig weiterentwickelt wird. Die aktuelle Version (zu Beginn dieser Arbeit) ist in der Abbildung 4 zu sehen. Sie unterstützt mehrere Filterfunktionen, nach denen die Menge der zur Auswahl stehenden POIs schrittweise eingeschränkt werden kann, auch fassetierte Suche [English u. a., 2002; Eyal Oren and Renaud Delbru and Stefan Decker, 2006] genannt. Aktuell können die POIs nach deren Lage (Provinz → Distrikt → Stadt), Klassifizierung und über eine Landkarte gefiltert werden. Als Teil dieser Arbeit soll außerdem eine Filtermöglichkeit für die touristischen Merkmale hinzukommen. Diesbezüglich sei auf die Abschnitte 5.4 und 8.3 verwiesen. Möchte ein Benutzer mehr zu einem bestimmten POI wissen, so werden ihm alle Informationen zu diesem POI auf einer speziellen Detailseite angezeigt (siehe Abbildung 5).

3.2 ETOURISMUS-VOKABULARE

Um die mannigfaltigen Informationen der eTourismus-Domäne abbilden zu können, kommen eine Reihe von Vokabularen, zum größten Teil als Eigententwicklungen, innerhalb des Vakantieland-Projekts zum Einsatz. Zur allgemeinen Beschreibung eines POIs werden die beiden Prädikate `dc:title` und `dc:description` aus dem Dublin Core Metadata-Vokabular [DCMI, 2008] verwendet. Für die Definition der Klassifikations- und Merkmals-Modelle wurden Elemente aus dem OWL-Vokabular verwendet. Das Listing 3 zeigt zwei der im Vakantieland-Projekt verwendeten Namensräume, welche im weiteren Verlauf dieser Arbeit verwendet werden. Das zentrale Element des Klassifikations-Modells stellt dabei die Klasse `etc:POI` dar. Bis zu dieser Arbeit wurden mehr als 400 weitere Subklassen dieser POI-Klasse

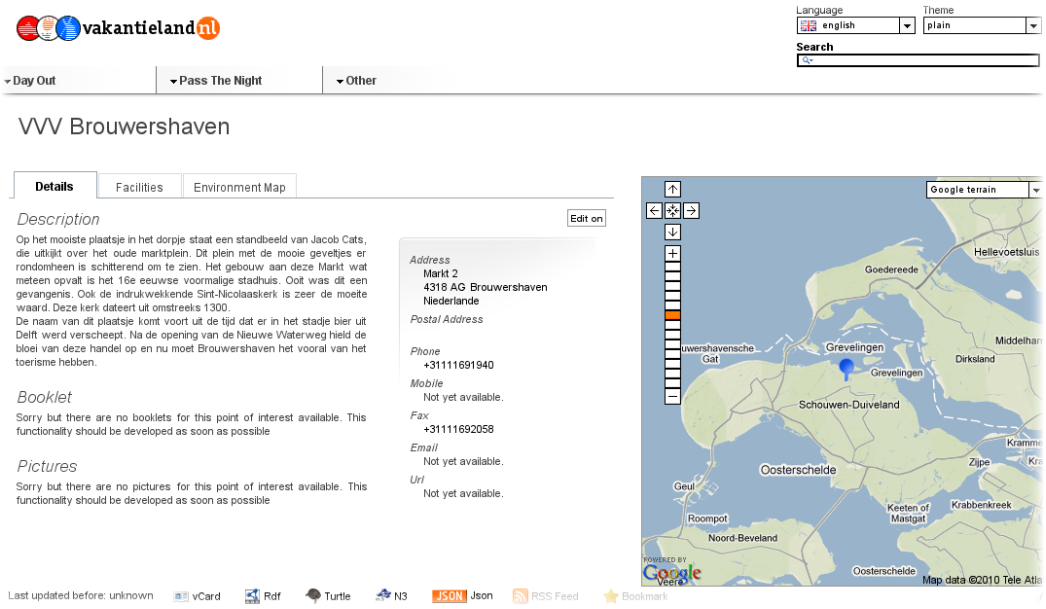


Abbildung 5: Detailseite eines POI in der Vakantieland-Applikation.

definiert, wie etwa `etc:Hotel`, `etc:Museum` oder `etc:Restaurant`. Für das Merkmals-Modell ist das Prädikat `etf:feature` von zentraler Bedeutung. Es repräsentiert die Relation „Merkmal eines Point of Interests“ und hat als Definitionsbereich (`rdfs:domain`) die Klasse `etc:POI`. Somit können alle Subprädikate, welche konkrete touristische Merkmale repräsentieren, auch nur für touristische Ziele verwendet werden. Zu Beginn dieser Arbeit sind mehr als 1200 solcher Submerkmale von `etf:feature` in einer Folksonomie[2]-ähnlichen Hierarchie ausgearbeitet. Diese Hierarchie einer Evolution zu unterziehen, um Inkonsistenzen und aus Sicht der Ontologie-Entwicklung fehlerhafte Modellierungen zu beseitigen, ist Hauptbestandteil dieser Arbeit und wird in den folgenden Kapiteln genauer erläutert. In Hinsicht auf die eTourismus-Domäne ist es weiterhin nötig, POIs mit Adress- bzw. Kontaktinformationen ausstatten zu können. Dazu wurden ebenfalls eigene Vokabulare definiert, auf die an dieser Stelle aber nicht näher eingegangen werden soll, da sie für diese Arbeit nicht relevant sind. Die Abbildung 6 zeigt als Beispiel die Repräsentation eines POI in Form eines RDF-Graphen.

```
1 etc: <http://vakantieland.nl/catalogue/classes/>
2 etf: <http://vakantieland.nl/catalogue/features/>
```

Listing 3: Auszug aus den definierten Namensräumen, welche innerhalb des Vakantieland-Projekts eingesetzt werden.

2 freie Verschlagwortung ohne Verwendung eines kontrollierten Vokabulars

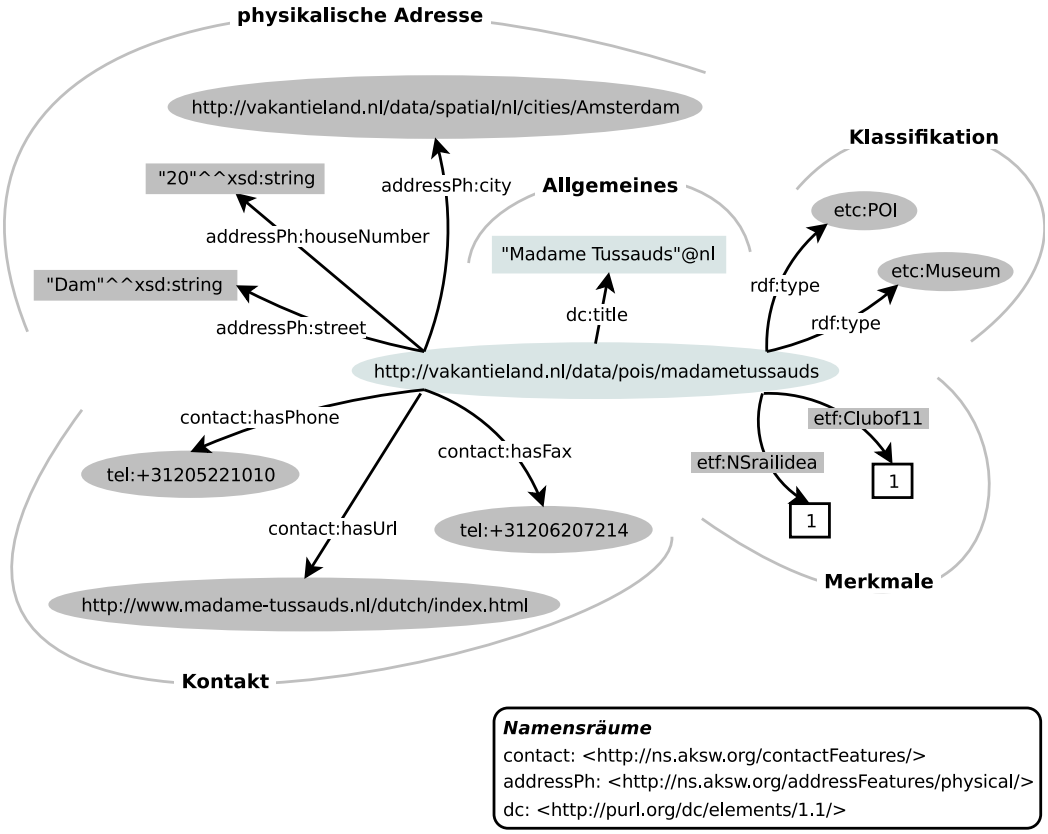


Abbildung 6: Graphische RDF-Repräsentation eines beispielhaften POI.

4 STAND DER TECHNIK

Inhaltsangabe

In diesem Kapitel werden Techniken und Werkzeuge zur Evolution einer Ontologie vorgestellt. Zu Beginn wird der Begriff der Ontologie-Evolution und der damit verbundene Evolutionsprozess beschrieben, welcher im Zuge dieser Arbeit angewandt wird. Anschließend werden unterschiedliche Möglichkeiten zur Darstellung von Änderungen an einer Ontologie diskutiert. Abschließend wird aus dieser Diskussion heraus ein Werkzeug vorgestellt, dass bei der Evolution des Schemas der touristischen Merkmale zum Einsatz kommen soll.

4.1 ONTOLOGIE-EVOLUTION

Nutzt eine Applikation zur Konzeptionalisierung und Speicherung seiner Daten eine Ontologie, so kann es durch sich ändernde Anforderungen vorkommen, dass eine alleinige Anpassung der Applikation nicht mehr ausreicht. Manche Anforderungen lassen sich nur durch eine Änderung der zugrunde liegenden Ontologie realisieren. Eine solche Ontologie-Evolution wird in [Stojanovic u. a., 2002] vorgestellt und wie folgt definiert:

> Ontology Evolution is the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology-based application, as well as the consistent management/propagation of these changes to dependent elements.

Es handelt sich demnach um eine zeitliche Anpassung einer Ontologie an geänderte Geschäftsanforderungen, sowie die Verwaltung und Weiterleitung dieser Änderungen an die abhängigen Artefakte.

4.1.1 Ontologie-Evolutionsprozess

Ähnlich einem Entwicklungsmodell aus der Software-Entwicklung stellt eine Ontologie-Evolution ebenfalls einen Prozess aus mehreren Phasen dar. Dieser Evolutionsprozess kann in sechs Phasen eingeteilt werden. Die Abbildung 7 zeigt, wie diese Phasen zeitlich miteinander in Zusammenhang stehen.

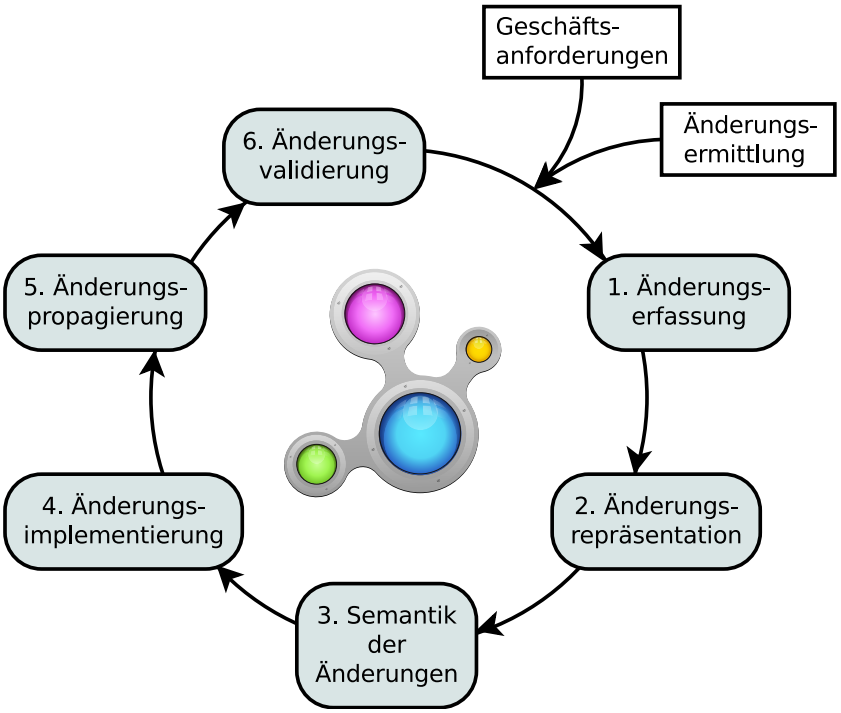


Abbildung 7: Evolutionsprozess einer Ontologie.

1. PHASE – ÄNDERUNGSERFASSUNG In dieser Phase wird entschieden, welche Anpassungen an der Ontologie durchgeführt werden sollen. Dies kann entweder durch explizite Anforderungen (zum Beispiel durch den Ontologie-Entwickler) oder durch automatische Ermittlungsverfahren ausgelöst werden.

2. PHASE – ÄNDERUNGSREPRÄSENTATION Um Änderungen nachvollziehen zu können, müssen diese identifiziert und in einem geeigneten Format dargestellt werden.

3. PHASE – SEMANTIK DER ÄNDERUNGEN Um die Konsistenz der Ontologie zu bewahren, müssen die Auswirkungen einer Änderung genau verstanden werden.

4. PHASE – ÄNDERUNGSIMPLEMENTIERUNG Bevor eine Änderung durchgeführt wird, müssen dem Nutzer alle Auswirkungen angezeigt werden,

die er entweder akzeptieren oder ablehnen kann. Erst wenn der Nutzer zustimmt, sollten die Änderungen ausgeführt werden.

5. PHASE – ÄNDERUNGSPROPAGIERUNG Sämtliche Artefakte, welche von den Änderungen betroffen sind, müssen in dieser Phase behandelt und wenn nötig angepasst werden.

6. PHASE – ÄNDERUNGSVALIDIERUNG In der letzten Phase müssen alle Änderungen validiert werden. Bei Bedarf muss es möglich sein, bestimmte Änderungen wieder rückgängig machen zu können.

Zusätzlich zu diesen Phasen gibt es noch vorbereitende Tätigkeiten, welche von außen heraus die Geschäftsanforderungen und die Ergebnisse der Änderungsermittlung in diesen Kreislauf hineinbringen. Die Änderungsermittlung kann hierbei in drei unterschiedliche Methoden eingeteilt werden [Stojanovic u. Stojanovic, 2002]. Einerseits lassen sich mögliche Verbesserungen durch die Analyse der Struktur der Ontologie und Anwendung verschiedener Heuristiken ermitteln, was als *Struktur-getriebene* Änderungsermittlung bezeichnet wird. So sollte etwa eine Klasse mit nur einer einzigen Subklasse mit dieser zusammengeführt werden. Bei der *Daten-getriebenen* Änderungsermittlung werden Anpassungen in der Domäne durch die Analyse der Instanzdaten bestimmt. Eine Klasse, welche keine Instanzen besitzt, stellt einen potenziellen Kandidaten für eine eventuelle Löschung dar. Die *Anwendungs-getriebene* Änderungsermittlung berücksichtigt die Nutzung der Ontologie. Sie basiert auf der Analyse des Nutzungsverhaltens der Anwender. Durch eine Anfrageverfolgung können zum Beispiel die am wenigsten oder gar nicht genutzten Ressourcen ausfindig gemacht werden, welche dann möglicherweise gelöscht oder angepasst werden sollten.

Änderungsermittlung

4.1.2 Evolutionsstrategie

Von Beginn an müssen während der Durchführung des Evolutionsprozesses Entscheidungen getroffen werden. Eine Evolutionsstrategie (laut [Stojanovic u. a., 2002] „advanced evolution stragety“ genannt) soll dafür sorgen, dass jede einzelne dieser Entscheidungen einheitlich nach festgelegten Vorgaben beschlossen wird. Dabei besteht eine solche Strategie aus mehreren Elementarstrategien, welche in den einzelnen Situationen für das weitere Vorgehen entscheidend sind [Stojanovic u. Motik, 2002]. So zum Beispiel die Frage, wie mit nicht mehr benötigten Ressourcen umgegangen werden soll oder die erlaubte Form einer Klassen- beziehungsweise Prädikathierarchie. Allgemein wird zwischen den folgenden Evolutionsstrategien unterschieden:

STRUKTUR-GETRIEBENE STRATEGIE: Die Änderungen werden entsprechend den Kriterien, wie die resultierende Ontologie auszusehen

hat, durchgeführt. Diese Strategie folgt den Anforderungen von Ontologie-basierten Applikationen.

PROZESS-GETRIEBENE STRATEGIE: Der Änderungsprozess selbst bestimmt, welche Änderungen wie durchgeführt werden sollen. Beispielsweise optimiert nach den Kosten oder der Anzahl von nötigen Schritten für eine Änderung.

INSTANZ-GETRIEBENE STRATEGIE: Die Änderungen werden so durchgeführt, dass ein bestimmter Zustand bei den Instanzen erreicht wird.

FREQUENZ-GETRIEBENE STRATEGIE: Die am häufigsten oder zuletzt verwendete Elementarstrategie wird angewandt.

4.2 DARSTELLUNG VON ONTOLOGIE-ÄNDERUNGEN

Das Ziel dieser Arbeit ist die Erzeugung einer neuen Version der verwendeten Ontologie mit einem angepassten Schema für die touristischen Merkmale. Bisher wird für jede neu installierte Vakantieland-Instanz die Ontologie durch eine semantische Annotation der alten relationalen Daten mit Hilfe eines Programms erstellt. Dieses Programm soll nicht verändert werden, wodurch die Angleichung der Ontologie an das neue Schema nach der semantischen Annotation stattfinden muss. Diese Anpassung der Ontologie kann nun auf verschiedenen Wegen dargestellt werden. Der erste und einfachste Weg um Änderungen festzuhalten, ist die Erzeugung eines Protokolls, in dem alle Änderungen in der exakten Sequenz ihrer Durchführung enthalten sind. Einige Ontologie-Entwicklungswerkzeuge unterstützen eine solche Erstellung eines Änderungsprotokolls, wie etwa der quelloffene und weit verbreitet genutzte Ontologie-Editor Protégé [Gennari u. a., 2003]. Das Problem bei der Verwendung eines solchen Protokolls besteht in der Erzeugung des selbigen. Für die eTourismus-Ontologie müsste eine manuelle Anpassung des Schemas und der Instanzdaten zu über 20.000 POIs durchgeführt werden. Dieser Aufwand liegt nicht im Sinne dieser Arbeit, wodurch diese Variante nicht in Frage kommt. Durch die Autoren [Klein u. Noy, 2003] werden drei weitere Varianten vorgestellt: (i) Bestimmung des strukturellen Unterschieds; (ii) Darstellung des Unterschieds in Form von konzeptuellen Änderungen oder (iii) in Form von Transformationsmengen.
Der erste Weg besteht aus der Bestimmung eines strukturellen Unterschieds zwischen den beiden Versionen der Ontologie. Für einen solchen Vergleich wurde ein spezieller Algorithmus namens PROMPTDIFF [Noy u. a., 2004] entwickelt, der als Plugin für Protégé zur Verfügung steht. Doch auch hier besteht das Problem, dass zunächst sämtliche Anpassung einmalig manuell ausgeführt werden müssten, um eine neue Version der Ontologie zu erhalten.

In der zweiten Variante werden alle Unterschiede zwischen den zwei Versionen einer Ontologie durch konzeptuelle Änderungen dargestellt. Zum Beispiel enthält eine konzeptuelle Änderung die Aussage, dass das Konzept `A` in der alten Version Subkonzept von `B` war, bevor es in der neuen Version verschoben wurde. An dieser Stelle könnte man nun die konzeptuellen Änderungen erstellen, ohne eine neue Version der Ontologie erzeugen zu müssen, insofern vorher genau definiert wurde, wie die neue Ontologie auszusehen hat. Für diesen Ansatz gibt es jedoch nach bisherigen Erkenntnissen keine Spezifikation oder unterstützende Werkzeuge.
Der dritte und letzte Weg besteht aus der Erzeugung von Transformationsmengen, welche sämtliche Operationen enthalten, die nötig sind, um eine Ontologie in eine neue Form zu überführen. Auch hier besteht der Vorteil, dass man die Transformationsmengen bestimmen kann, ohne vorher mühsam die neue Version der Ontologie erzeugen zu müssen. Die endgültige Struktur muss jedoch ebenfalls vorher definiert sein. Ein Werkzeug, welches die Erstellung solcher Transformationsmengen in Form von Evolutionsmustern unterstützt, soll nun im folgenden Abschnitt vorgestellt werden.

4.3 ONTOWIKI

Bei OntoWiki handelt es sich um ein semantisches Datenwiki zur kollaborativen Erstellung und Veröffentlichung von RDF-Wissensbasen [Tramp u. a., 2010a]. Innerhalb des Vakantieland-Projekts wird es bereits zur Verwaltung und Wartung der eTourismus-Ontologie verwendet. OntoWiki bietet in einer Web-basierten Umgebung Funktionen zur gemeinsamen Verwaltung von strukturierten Informationen. Dabei stehen den Autoren ausgeklügelte Mechanismen zur Navigation, Visualisierung und Bearbeitung zur Verfügung. So kann entweder über die Taxonomie beziehungsweise Hierarchie, mit Hilfe einer fassetierten Suche oder durch eine Volltext-Suche durch eine Wissensbasis navigiert werden. Die Ergebnisse einer solchen Suche werden standardmäßig in Tabellenform angezeigt. Durch ein flexibles Erweiterungssystem können aber zusätzliche, spezielle Ansichten implementiert werden, zum Beispiel Karten oder Kalender. Auch die Ansicht einer Ressource (siehe Abbildung 8) kann mit Hilfe des Erweiterungssystems angepasst und bei Bedarf spezialisiert werden. Alle diese Ansichten können mit RDFa [Adida u. a., 2008] annotiert und mittels RDFauthor [Tramp u. a., 2010b], einem JavaScript-basierten Werkzeug zur Verarbeitung von RDF, editiert werden. Wie bei einem Wiki üblich, besitzt OntoWiki ebenfalls eine Versions-Kontrolle, wodurch jede Änderung rückgängig gemacht werden kann.

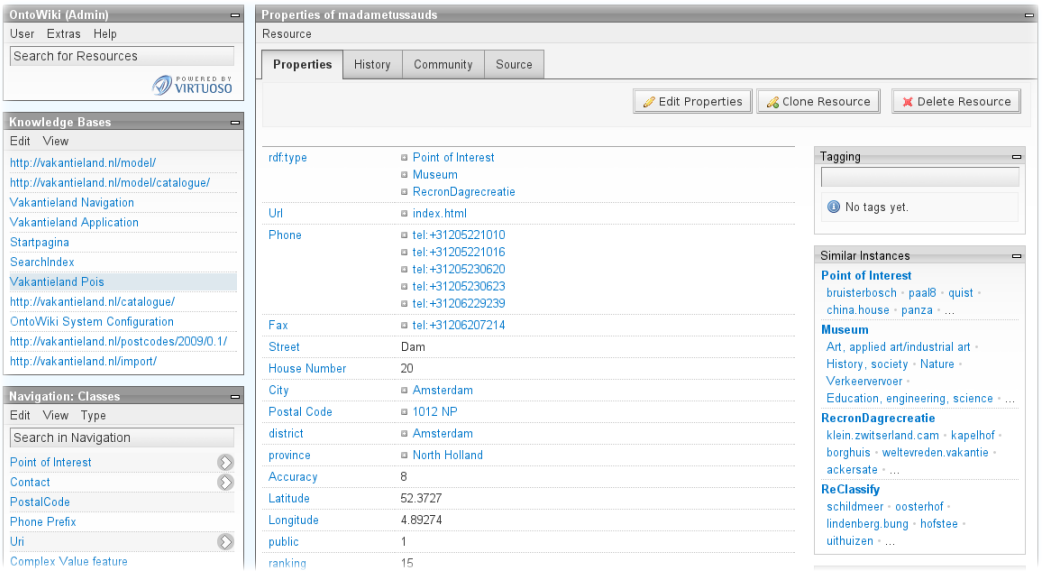


Abbildung 8: Ansicht einer RDF-Ressource in OntoWiki.

4.3.1 Evolutionskomponente

Wie bereits im Abschnitt 4.1 erläutert, kommt es bei der Nutzung von Ontologien immer wieder dazu, dass Anpassung an diesen nötig sind. Dazu wurde eine Komponente für OntoWiki entwickelt [Rieß, 2010], mit deren Hilfe sich allgemeine, vordefinierte Evolutionsmuster ausführen lassen. Ein solches Evolutionsmuster stellt eine Art Transformationsmenge mit den benötigten Operationen für eine gewünschte Anpassung dar. Zusätzlich können auch eigene Muster erstellt werden, um so spezielle Evolutionsszenarien zu verwirklichen. In der Abbildung 9 ist die Ansicht zur Bearbeitung eines solchen Musters zu sehen.

Evolutionsmuster

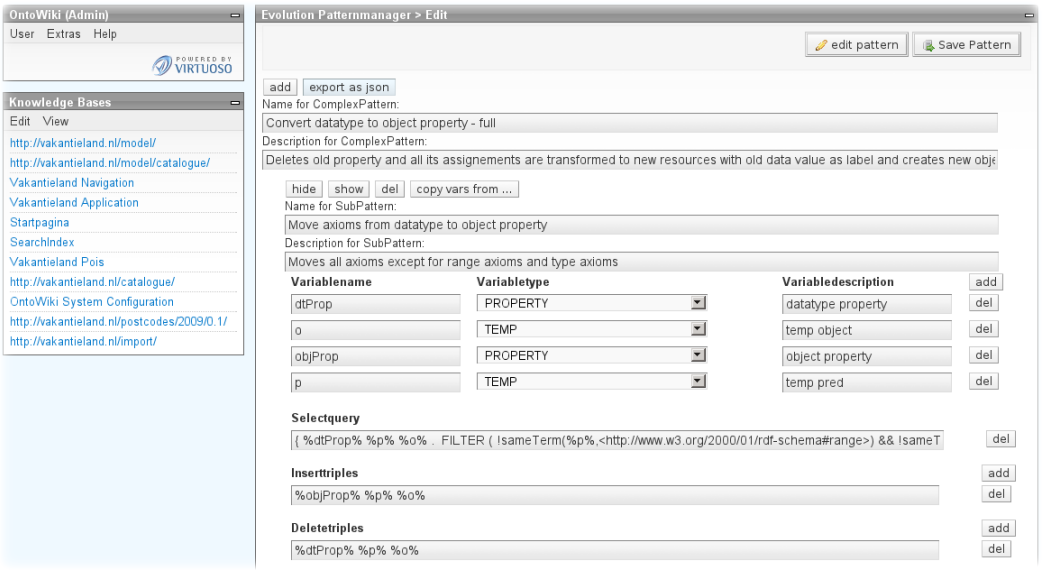


Abbildung 9: Bearbeitungsansicht eines Evolutionsmusters in OntoWiki.

Ein Evolutionsmuster besteht aus einem `Compound Evolution Pattern` (CP), welches wiederum aus einer Sequenz von mindestens einem oder mehreren `Basic Evolution Pattern` (BP) zusammengesetzt ist [Rieß u. a., 2010]. CPs sowie BPs besitzen einen Bezeichner (`label`) und eine Beschreibung (`desc`). Ein Basic Evolution Pattern enthält zudem ein Tupel aus einer Menge von Variablen `V`, die `WHERE`-Klausel einer SPARQL-Anfrage `S` mit Platzhaltern für die Variablen aus `V` und einer Menge von SPARQL/Update-Anfragen [Seaborne u. a., 2008] `U`, ebenfalls mit Platzhaltern für Variablen aus `V`. Ein Platzhalter wird durch ein „%“-Zeichen vor und hinter dem Variablennamen definiert. Zur Laufzeit werden diese Platzhalter entweder durch einen dafür vorgegebenen Wert gebunden oder durch eine SPARQL-Variable ersetzt.

```
1  {
2   "label":"replaceLabel",
3   "desc":"Replaces the label of a resource.",
4   "subPattern":[{
5     "label":"replaceLabelOfResource",
6     "desc":"Replaces the label of a given resource.",
7     "V":[
8       {"name":"res","bound":true,"type":"RESOURCE","desc":""},
9       {"name":"newLabel","bound":true,"type":"LITERAL","desc":""},
10      {"name":"lang","bound":true,"type":"LITERAL","desc":""},
11      {"name":"label","bound":false,"type":"TEMP","desc":""}
12    ],
13    "S":"{ %res% rdf:label %label% FILTER
          langMatches(lang(%label%),%lang%) }",
14    "U":[
15      {"pattern":"%res% rdf:label setLang(%newLabel%,%lang%)",
16       "type":"insert"},
17      {"pattern":"%res% rdf:label %label%",
18       "type":"delete"}
19    ]
20  }]
21 }
```

Listing 4: Beispiel eines Evolutionsmusters in JSON.

Das Listing 4 zeigt ein Evolutionsmuster im JSON[1]-Format. In den Zeilen 8-11 werden die Variablen definiert. Variablen vom Typ `TEMP` werden dabei durch die `WHERE`-Klausel `S` (Zeile 13) gebunden. Alle anderen Variablentypen bekommen ihren Wert von außen durch das System bzw. den Benutzer zugewiesen. Die Menge `U` (Zeile 14-18) besteht in diesem Beispiel aus zwei SPARQL/Update-Anfragen. Zum einen aus einer `Insert`-Anfrage, welches für jedes Tupel aus der Ergebnismenge von `S` ein neues Tripel entsprechend dem angegebenen Tripel-Muster erzeugt. Zum anderen aus einer `Delete`-Anfrage, welche ebenfalls für jedes Tupel aus dem Ergebnis von `S` entsprechende Tripel aus der Datenbank löscht. Die Evolutionskomponente besitzt außerdem einen Präprozesser zur Ausführung

1 http://www.json.org/

bestimmter, vordefinierter Funktionen. In Zeile 15 wird eine solche Funktion, `setLang()`, zum Setzen der Sprachangabe für ein ungetyptes Literal verwendet.

5 ANFORDERUNGSANALYSE

Inhaltsangabe

In diesem Kapitel werden die Anforderungen an ein Schema erarbeitet, welches eine einheitliche und erweiterbare Modellierung von touristischen Merkmalen in RDF ermöglicht. Für den Anwendungsfall der Integration einer fassetierten Suche auf Basis dieses Schemas in die Applikation Vakantieland, sowie für das Programm zur Durchführung der Evolution des alten Merkmals-Schemas werden ebenfalls Anforderungen definiert. Diesbezüglich werden zu Beginn Anwendungsszenarien für alle drei Teilbereiche vorgestellt, um ein möglichst breites Spektrum von benötigten Charakteristika und Funktionen abzudecken.

5.1 ANWENDUNGSSZENARIEN

In den folgenden Szenarien wird zwischen zwei verschiedenen Anwendergruppen unterschieden. Zum einen die Gruppe bestehend aus Administratoren, welche für die Verwaltung der Daten innerhalb der eTourismus-Ontologie verantwortlich sind. Zum anderen die Anwendergruppe der Touristen, welche sich über die Vakantieland-Applikation Informationen zu den touristischen Zielen einholen wollen.

5.1.1 Wartung der eTourismus-Ontologie

Im Falle des Vakantieland-Projekts wird eine OntoWiki-Instanz zur Verwaltung der eTourismus-Ontologie verwendet. Bisherige Erfahrungen haben aber gezeigt, dass die aktuelle Modellierung der touristischen

Merkmale und deren Hierarchie es verhindern, die Möglichkeiten von OntoWiki für eine effektive Wartung auszuschöpfen. Dieses Szenario soll nun zeigen, wie eine bessere Verwaltung unter der Nutzung der Funktionalität von OntoWiki aussehen könnte.

Ein Administrator möchte einen neuen POI anlegen. Dazu meldet er sich in der OntoWiki-Applikation an und wählt den entsprechenden Instanz-Graphen aus, in dem sämtliche POIs gespeichert sind. In der Navigationsbox erscheint nun die Klasse „Point Of Interest“. Mit einem Klick auf den Knopf zur Anzeige weiterer Optionen für diese Klasse wählt er die Option „Instanz erstellen“. Nun erscheint ein Dialog (siehe Abbildung 10) zur Erzeugung einer neuen Ressource vom Typ `etc:POI`.

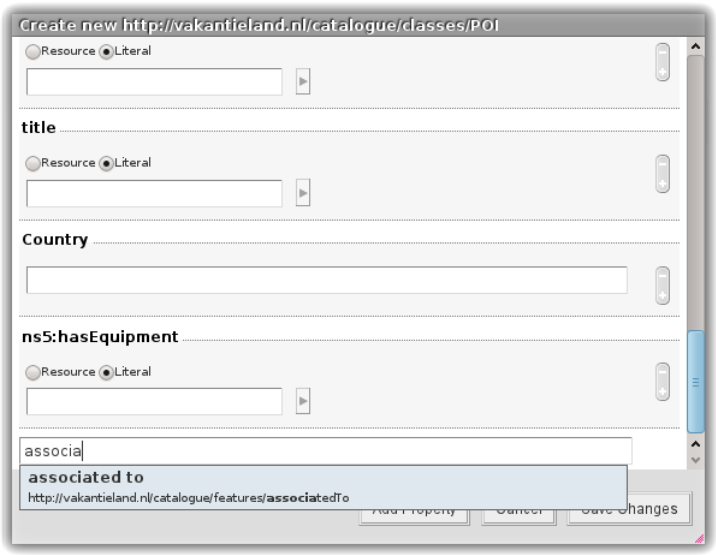


Abbildung 10: Dialog zur Erzeugung einer neuen Ressource in OntoWiki.

Dort können Angaben zur Adresse, zu Kontaktdaten, zur allgemeinen Beschreibung, zur Klassifikation und eben auch zu den Merkmalen des POI gemacht werden. Um ein Merkmal für den POI anzugeben, drückt der Administrator auf die Schaltfläche „Neues Prädikat“. In einem neuen Eingabefeld mit Autovervollständigung kann er das entsprechende Merkmal auswählen. Da für jedes Merkmal ein Wertebereich angegeben ist, kann er direkt aus einer Liste den Wert für das Merkmal auswählen. Während der Erstellung fällt ihm auf, dass es noch kein Merkmal zur Angabe von Raumausstattungen gibt. Damit er die Erstellung des POIs nicht unterbrechen muss, kann er sofort, entsprechend den Vorgaben eines Entwurfsmusters, eine neue URI in das Textfeld für das Prädikat eingeben. Für den Wert gibt er nun, auch wieder nach der Definition des Musters, entweder eine URI für eine Ressource oder ein Literal an. Sind alle Angaben zu dem POI vollständig, kann er den Dialog schließen, indem er auf „Speichern“ klickt. Allerdings befindet sich die Ontologie jetzt in einem nicht konsistenten Zustand, da das Entwurfsmuster für das Merkmal noch nicht komplett umgesetzt ist. Um die Definition des Merkmals zu vervollständigen,

wählt er dieses aus der Navigationsbox aus, nachdem er dort auf die Prädikatsansicht gewechselt hat. Nun öffnet sich die Ressourcen-Ansicht, wo er alle Vorgaben des entsprechenden Entwurfsmusters umsetzt. Danach befindet sich die Ontologie zwar wieder in einem konsistenten Zustand, jedoch handelt es sich bei einem Merkmal um ein Schema-Element und sollte deswegen auch im Schema-Graphen gespeichert werden. Zum Zeitpunkt der Erstellung dieses Merkmals war aber der Instanz-Graph ausgewählt. Aus diesem Grund wechselt der Administrator in die Komponente zur Evolution und verschiebt mit Hilfe eines Evolutionsmusters das Merkmal in den Schema-Graphen.

5.1.2 Evolution der eTourismus-Ontologie

Eine externe Anforderung legt fest, dass die Änderungen im Schema für die touristischen Merkmale und somit die Evolution der eTourismus-Ontologie während des Installationsprozesses einer Vakantieland-Instanz ausgeführt werden soll. Auf der Basis dieser Anforderung könnte ein mögliches Programm zur Durchführung einer Evolution wie folgt aussehen:

Die Daten der eTourismus-Ontologie werden aktuell in einer speziellen Datenbank für RDF-Daten, im Allgemeinen als Triple-Store bezeichnet, gespeichert. Im konkreten Fall des Vakantieland-Projekts handelt es sich um die Open-Source Variante des universellen Datenbank-Servers Virtuoso von OpenLink Software[1]. Diese Datenbank unterstützt das Anfragen der gespeicherten Daten, sowie das Einfügen und Löschen von RDF-Tripeln. Daraus ergibt sich der folgende, stark abstrahierte Algorithmus zur Anpassung der Datenbasis:

SELEKTION Durch eine Anfrage können bestimmte Informationen aus der Datenbank selektiert werden.

EINFÜGEN Mit Hilfe der Ergebnismenge aus der Selektion und von außen eingebrachten Parametern, wird eine Menge von Tripeln erstellt, welche in die Datenbank eingefügt werden sollen. Dabei stehen Funktionen zur Verfügung, um Manipulationen an den Werten aus der Ergebnismenge und den Parametern durchführen zu können.

LÖSCHEN Ähnlich wie beim Einfügen, wird eine Menge von Tripeln erstellt, welche aus der Datenbank entfernt werden sollen. Dazu stehen ebenfalls die Ergebnismenge aus der Selektion, die Parameter sowie Funktionen zur Manipulation zur Verfügung.

Alle drei Schritte sind dabei optional, jedoch muss mindestens ein Tripel entweder eingefügt oder gelöscht werden, damit man von einer Anpas-

1 http://virtuoso.openlinksw.com/

sung der Datenbasis sprechen kann. Eine Anleitung, welche für den Algorithmus vorgibt, was jeweils in den drei Schritten zu tun ist, soll an dieser Stelle als Rezept bezeichnet werden. Ein Rezept dient also zur Anpassung von Ressourcen. Dabei sollte ein Rezept möglichst effizient sein, das heißt so viel wie nötig und so wenig wie möglich Operationen, um die gewünschte Anpassung herbeizuführen. Dadurch lässt sich ein Rezept besser nachvollziehen, wodurch die dritte Phase des Evolutionsprozesses, die Semantik der Änderungen, unterstützt wird. Da die Evolution der Ontologie innerhalb des Installationsprozesses und vollautomatisch (ohne Nutzerinteraktion während der Ausführung) zu realisieren ist, muss die Möglichkeit bestehen, verschiedene solcher Rezepte durch ein Programm hintereinander ausführen zu können. Mit Hilfe einer Konfiguration sollte für dieses Programm festgelegt werden können, wann für welche Ressource welches Rezept angewandt werden soll. Dies hilft in der sechsten Phase des Evolutionsprozesses, der Änderungsvalidierung, bestimmte Anpassungen eventuell auszulassen beziehungsweise abzuändern. Bei einem Fehler während der Ausführung des Evolutionsprogramms soll dieses sofort stoppen und eine geeignete Meldung ausgeben.

5.1.3 Reiseplanung mit Vakantieland

In diesem Abschnitt soll ein fiktives Szenario vorgestellt werden, wie Touristen sich über die Vakantieland-Applikation Informationen zu möglichen POIs einholen können.

Ein Tourist wurde durch seine Freunde dazu beauftragt, eine mehrtägige Radtour zu planen. Dazu nimmt er sich seinen Laptop und wählt bequem vom Sofa aus die Internetseite des ihm empfohlenen Tourismus-Portals Vakantieland an. Dieses ermöglicht es ihm, ohne besondere Vorkenntnisse aus einer zuvor unüberschaubaren Menge, die für ihn relevanten POIs herauszusuchen. Dazu werden ihm eine Reihe unterschiedlicher Filter- und Suchfunktionen geboten. Da sich die Freunde zuvor ausgemacht hatten, dass die Radtour durch eine bestimmte Region des Landes führen soll, wählt er nun zuerst die Provinz aus, in der die Region liegt. Als nächstes möchte er mögliche POIs zum Übernachten heraussuchen. Um Geld zu sparen, wollen die Freunde die Nächte jeweils auf einem Campingplatz mit Möglichkeit zum Zelten verbringen. Daher selektiert er die Kategorie „Campingplatz“ und wählt zudem „Zeltstellplätze“ des Merkmals „Campingart“. Weil die Trefferliste immer noch zu umfangreich ist, sucht er in der Merkmals-Filterbox nach einem weiteren Kriterium. Er stößt auf das Merkmal „Ausstattung“ und klickt auf den Wert „Supermarkt“, hinter dem eine „10“ für die Anzahl der zutreffenden POIs steht. Durch diese Auswahl, denkt er sich, könnten die Freunde morgens, vor der Weiterfahrt, Proviant einkaufen. Auf der Karte, welche ebenfalls eingeblendet ist, kann er sich direkt die Lage der gefundenen POIs anschauen. Er ist zufrieden mit den Standorten und schaut sich nun durch jeweiliges Anklicken

der Treffer die Detailseiten zu den POIs an. Dort kann er Beschreibungen, Bilder und Kontaktinformationen einsehen, sowie eine komplette Liste aller Merkmale zu dem jeweiligen POI. Mit den Navigationsschaltflächen des Browsers kann er jederzeit wieder zurück zur Trefferliste. Da er die Entscheidung, auf welchen Campinplätze sie wann übernachten wollen, nicht selber treffen will, kopiert er die URL aus der Adresszeile des Browsers und schickt diesen Link per Email an seine Freunde. Diese gelangen durch einen Aufruf der URL im Browser zu genau der gleichen Trefferliste wie ihr Freund und können sich die Kriterien anschauen, nach denen er gefiltert hat.

5.2 SCHEMA FÜR TOURISTISCHE MERKMALE

Das Schema soll eine geeignete Modellierung von touristischen Merkmalen und deren touristischen Werten ermöglichen. Geeignet bedeutet in diesem Zusammenhang, dass es den Anforderungen aus dem Anwendungsszenario 5.1.1 genügt, aber auch Erfahrungen aus dem Einsatz der Ontologie innerhalb der Vakantieland-Applikation berücksichtigt werden. Zwei relevante Erfahrungen sollen an dieser Stelle kurz erläutert werden:

- Die Performanz ist eine der schwierigsten Hürden beim Einsatz eines Triple-Stores zur Persistierung der Daten in einer semantischen Web-Applikation [Martin u. a., 2010]. Aus diesem Grund sollten so wenig wie möglich Anfragen an die Datenbank gestellt werden. Auch spielt die Komplexität einer Anfrage eine wesentliche Rolle. So haben die Anzahl an Tripel-Mustern, die Verwendung von Werte-Prüfungen und auch die Größe der Ergebnismenge einen entscheidenden Einfluss auf die Antwortzeit einer Anfrage.
- Wie schon im Grundlagen-Kapitel erläutert (siehe Abschnitt 2.3), ist der Einsatz von anonymen Knoten problematisch. Da auch in der Vakantieland-Applikation SPARQL zur Anfrage der Daten verwendet wird, sollten anonyme Knoten bei der Modellierung vermieden werden.

Da es sich um ein Schema handelt, wird nicht zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden.

5.2.1 Anforderungen

UNTERSTÜTZUNG FÜR `rdfs:range` Das Schema muss die Einschränkung des Wertebereichs eines Merkmals mittels `rdfs:range` bieten. Dies resultiert bei der Wartung in einer leichteren Auswahl der möglichen Werte, da das Merkmal nur auf Instanzen einer bestimmten Klasse beschränkt ist. So zeigen viele Ontologie-Editoren, wie auch OntoWiki, bei der Wertezuweisung nur den eingeschränkten Wertebereich an. *A010*

UNTERSTÜTZUNG FÜR `rdfs:domain` Das Schema muss die Einschränkung des Defintionsbereichs eines Merkmals mittels `rdfs:domain` bieten. Da das Wurzelmerkmal `etf:feature` auf die Klasse `etc:POI` beschränkt ist, gilt dies ebenfalls für alle Subprädikate. Jedoch werden manche Merkmale nur durch einen Typ von POIs verwendet und sollten daher auch nur durch diesen Typ gebunden werden können. Diese Maßnahme erhöht wiederum die Wartbarkeit der POI-Instanzen und unterstützt spezielle Mechanismen in OntoWiki und anderen Editoren. *A020*

TRENNUNG ZWISCHEN SCHEMA- UND INSTANZDATEN Die eTourismus-Ontologie setzt sich aus mehreren Graphen zusammen, darunter einen ausschließlich für Schema-Daten und einen weiteren für sämtliche Instanz-Daten. Mittels `owl:imports` wird der Schema-Graph in den Instanz-Graph eingebunden. Diese Trennung soll durch das Merkmals-Schema aufrechterhalten werden. Die Definitionen der Merkmale sowie die Klassen für die Merkmalswerte (alle Tripel, in den die Merkmale und Werteklassen als Subjekt auftreten) werden somit im Schema-Graphen gespeichert. Die anderen Informationen, wie die Instanzen der Werteklassen, werden im Instanz-Graph hinterlegt. *A030*

EFFEKTIVE PERSISTIERUNG Die Modellierung eines Merkmals soll sowohl auf Schema- sowie bei der Verwendung dessen auf Instanz-Ebene so effektiv wie möglich gestaltet sein. Konkret bedeutet das, dass so wenig wie möglich Tripel erzeugt werden müssen, um die gewünschte Information abzubilden. Dies soll verhindern, dass die Anfragen an die Datenbank zu komplex werden, was wiederum die Performanz der gesamten Vakantieland-Applikation negativ beeinträchtigen kann. *A040*

DEFINITION VON ENTWURFSMUSTERN Die touristischen Merkmale lassen sich nicht alle auf die gleiche Art und Weise in RDF abbilden. So müssen Merkmale, welche Literale binden, anders modelliert werden, als welche, die URIs als Werte binden. Für diese verschiedenen Arten von Merkmalen sollen Entwurfsmuster definiert werden, welche die jeweilige Modellierung in RDF vorgeben. Dadurch soll ein einheitliches Schema gewährleistet werden, wodurch die Verarbeitung der Merkmale in darauf aufbauenden Artefakten, wie zum Beispiel der Vakantieland-Applikation, vereinfacht wird. *A050*

5.3 DURCHFÜHRUNG DER EVOLUTION

Im Abschnitt 4.3.1 wurde bereits ein System zur Evolution von Ontologien vorgestellt. Auf Basis des Szenarios 5.1.2 soll an dieser Stelle ermittelt werden, welche Anforderungen bereits durch dieses System erfüllt werden:

Wie im Szenario erwähnt, sollen bestimmte Anpassungen durch ein Rezept definiert werden können. Das Evolutionssystem von OntoWiki besitzt die Möglichkeit zur Definition und Ausführung selbsterstellter Evolutionsmuster, welche den Rezepten aus dem Szenario entsprechen. Es soll daher für die Ausführung der Evolution genutzt werden. Jedoch gibt es bisher keinen Mechanismus, um die Evolutionsmuster automatisch ausführen zu können. Es bedarf der manuellen Betätigung eines Administrators innerhalb der OntoWiki-Applikation. Da die Evolution des Merkmals-Schemas aber während des Installationsprozesses einer Vakantieland-Instanz und ohne Nutzerinteraktion stattfinden soll, muss eine Funktion zur Verfügung stehen, mit der durch einen entfernten Aufruf aus einem Programm heraus ein Evolutionsmuster ausgeführt werden kann. Für entfernte Aufrufe bietet OntoWiki eine RPC-Schnittstelle auf Basis von JSON. Bisher ist das Evolutionssystem aber noch nicht an diese Schnittstelle angebunden.
Aus dem Szenario und der eben geführten Vorüberlegung ergeben sich nun die folgenden Anforderungen.

5.3.1 Funktionale Anforderungen

RPC-SCHNITTSTELLE FÜR ONTOWIKI-EVOLUTIONSSYSTEM *FA010*

Um die Ausführung eines Evolutionsmusters aus einem externen Programm zu ermöglichen, muss eine RPC-Schnittstelle für das Evolutionssystem von OntoWiki implementiert werden. Dafür soll die bereits vorhandene `jsonrpc` OntoWiki-Komponente um eine Unterstützung für das Evolutionssystem erweitert werden. Da das Evolutionssystem zur Ausführung eines Evolutionsmusters neben dem Muster selbst, auch einen Standardgraphen und die Werte für die Variablen aus dem Muster benötigt, muss die Schnittstelle die Entgegennahme dieser Parameter ermöglichen. Bei unvollständigen oder fehlerhaften Parametern sowie bei Fehlern während der Ausführung soll eine entsprechende Fehlermeldung durch die Schnittstelle geworfen werden.

PROGRAMM ZUR AUSFÜHRUNG DER EVOLUTION *FA020*

Um die Evolution an der Ontologie durchzuführen muss ein Programm entwickelt werden. Dieses soll die definierten Evolutionsmuster aus *NFA010* schrittweise ausführen. Eine Konfiguration soll festlegen, welches Evolutionsmuster wann und mit welchen Parametern abgearbeitet wird. Zur Ausführung müssen die Evolutionsmuster und ihre Parameter an die in *FA010* erläuterte RPC-Schnittstelle übertragen werden. Wird eine Fehlermeldung zurückgegeben, soll die ausgegeben werden und das Programm bricht an dieser Stelle ab.

HTTP POST *FA021*

Die Daten werden an die `jsonrpc` OntoWiki-Komponente per HTTP POST [Fielding u. a., 1999] übertragen, um auch größere Datenmengen zu unterstützten. Aus diesem Grund

muss das Evolutionsprogramm eine Übertragung der Evolutionsmuster und deren Parameter über HTTP POST ermöglichen.

HTTP-AUTHENTIFIZIERUNG Das Evolutionssystem von OntoWiki kann nur mit bestimmten Rechten genutzt werden und verwendet für die Authentifizierung die Basic Authentication [Franks u. a., 1999] des HTTP-Protokolls. Eine Unterstützung für diese Art der Authentifizierung muss durch das Evolutionsprogramm gewährleistet werden. *FA022*

5.3.2 Nicht-funktionale Anforderungen

Allgemein soll für die Implementierung des Evolutionsprogramms die Programmiersprache PHP verwendet werden. Die Evolutionsmuster werden im JSON-Format notiert. Außer den beiden Anwendungen Vakantieland und OntoWiki sollen keine weiteren externen Programme oder Bibliotheken zur Ausführung des Evolutionsprogramms benötigt werden.

DEFINTION VON EVOLUTIONSMUSTERN Um das Schema der Merkmale an die neu definierten Entwurfsmuster aus *A050* anzupassen, müssen entsprechende Evolutionsmuster definiert werden, welche die alten Schema- und Instanzdaten in die neue Form überführen (zweite Phase des Evolutionsprozesses). Es sollen dabei so wenig wie möglich Operationen für die gewünschte Anpassung durchgeführt werden müssen. Dadurch können die Änderungen leichter nachvollzogen werden, wodurch wiederum die dritte Phase des Evolutionsprozesses unterstützt wird. *NFA010*

5.4 ANWENDUNGSFALL VAKANTIELAND

Die fünfte Phase des Evolutionsprozesses sieht eine Anpassung aller durch die Änderungen an der Ontologie betroffenen Artefakte vor. Aus diesem Grund müssen die entsprechenden Stellen in der Vakantieland-Applikation an das neue Merkmals-Schema angeglichen werden. Darüber hinaus sind neue Funktionen notwendig, um den Anforderungen aus dem Szenario 5.1.3 zu genügen.

5.4.1 Funktionale Anforderungen

ANPASSUNG AN NEUES MERKMALS-SCHEMA Zu den durch die Änderungen am Merkmals-Schema betroffenen Stellen gehört die Detailseite zur Anzeige aller Informationen zu einem bestimmten POI. Dort werden unter anderem auch die Merkmale des POIs in einer Liste angezeigt. Diese Liste muss an das neue Schema angepasst werden. *FA030*

FASSETIERTEN SUCHE FÜR MERKMALE *FA040*
Auf der Startseite der Vakantieland-Applikation stehen mehrere Filtermöglichkeiten zur Verfügung. Um den vollen Funktionsumfang, welcher im Szenario 5.1.3 beschrieben wird, zur Verfügung zu stellen, muss ein neuer Filter zur Einschränkung der angezeigten POIs nach deren Merkmalen implementiert werden. Dieser Filter soll sich nach den Prinzipien der fassetierten Suche richten.

VERKNÜPFUNG MIT BESTEHENDEN FILTERMECHANISMEN *FA041*
Damit der volle Möglichkeitsraum zur Filterung von POIs ausgeschöpft werden kann, muss die fassetierte Suche für die touristischen Merkmale mit den anderen Filterfunktionen (siehe Abschnitt 3.1) kombinierbar sein.

ZUSTANDSLOSIGKEIT *FA042*
Sämtliche ausgewählte Filterkriterien werden über HTTP GET [Fielding u. a., 1999] übertragen. Somit kann jede Kombination aus gewählten Filtern als Lesezeichen im Webbrowser abgespeichert beziehungsweise kann die URL kopiert und verteilt werden. Diese Funktionalität muss auch für die Filterkriterien aus der fassetierten Suche für die Merkmale garantiert werden.

MEHRFACHAUSWAHL *FA043*
Entsprechend dem Szenario soll ein Tourist in der Lage sein, mehrere Merkmale parallel auszuwählen. Außerdem muss die Möglichkeit bestehen, das selbe Merkmal mehrmals, aber mit unterschiedlichen Werten, insofern verfügbar, anwenden zu können.

6 ONTOLOGIE-ENTWURFSMUSTER

Inhaltsangabe

Das neue Schema für die touristischen Merkmale soll so weit wie möglich generisch gestaltet werden, um es auch in anderen Bereichen innerhalb der Tourismus-Domäne einsetzen zu können. Dennoch wird bei der Definition der Entwurfsmuster in erster Linie der Einsatz in der Vakantieland-Applikation berücksichtigt, welche als konkreter Anwendungsfall dient. Zu Beginn dieses Kapitels wird zunächst eine Analyse des alten Schemas durchgeführt. Die Erkenntnisse aus dieser Analyse bilden die Basis für eine Einteilung der Merkmale in unterschiedliche Problemfelder. Im Anschluss daran soll für jedes dieser Problemfelder zunächst geklärt werden, wie es für einen effektiven Einsatz korrekt in RDF modelliert werden kann. Dazu wird jeweils eine Diskussion geführt, aus deren Entscheidungen letztendlich ein Entwurfsmuster für das entsprechende Problemfeld abgeleitet und definiert wird.

6.1 ANALYSE DES ALTEN MERKMALS-SCHEMAS

Das alte Schema für die touristischen Merkmale besteht aus einer Hierarchie, welche mit Hilfe des RDFS-Elements `rdfs:subPropertyOf` aufgebaut ist. Die Wurzel dieser Hierarchie stellt das Prädikat `etf:feature` dar. Zu Erinnerung, es repräsentiert die Relation „Merkmal eines Point Of Interests“. Ausgehend von diesem Prädikat wurde eine Folksonomie aus touristischen Merkmalen erstellt. Die Abbildung 11 zeigt zur Veranschaulichung einen Ausschnitt aus dieser Hierarchie.
Darin ist auch zu sehen, dass mehrere Merkmale durch den selben Knoten, der ebenfalls ein Merkmal darstellt, zu einer Domäne zusammengefasst werden. Es besteht aber das Problem, dass nicht eindeutig

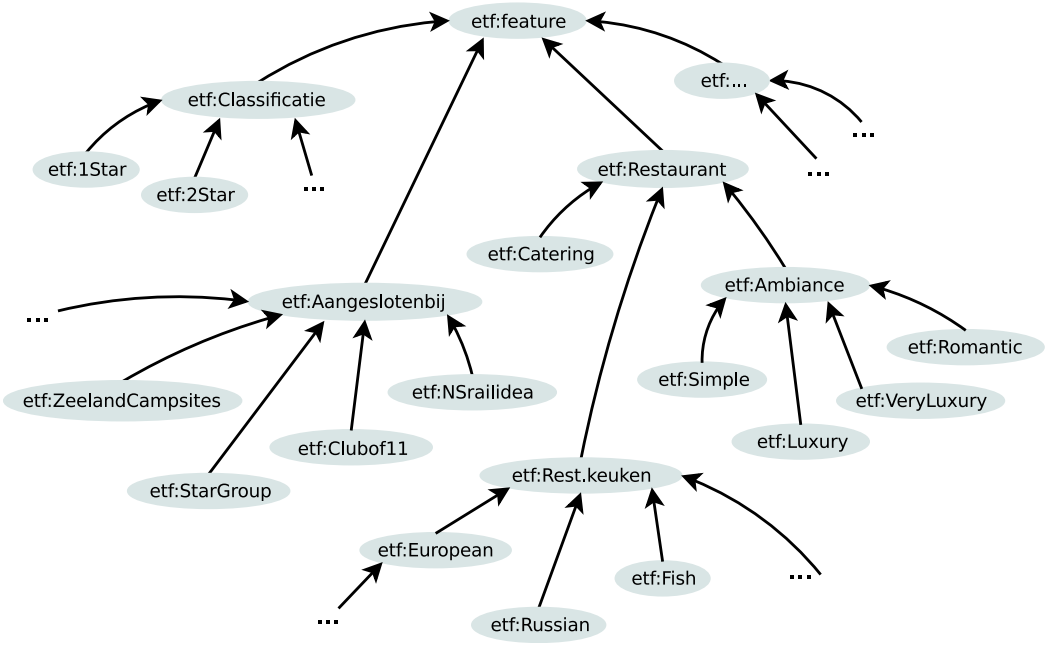


Abbildung 11: Ausschnitt aus der Hierarchie des alten Merkmals-Schemas. Die Merkmale sind untereinander durch `rdfs:subPropertyOf` verbunden.

festgelegt ist, ob ausschließlich Blätter oder auch Knoten dieser Hierarchie auf Instanz-Ebene zur Angabe von Merkmalen für POIs verwendet werden dürfen. Dies kann die Semantik eines Merkmals beeinflussen. So dient beispielsweise das Merkmal `etf:Rooms` einerseits zur Angabe der Anzahl von Räumen, wenn es mit einem POI verwendet wird. Andererseits besitzt es weitere Submerkmale wie etwa `etf:Bridalsuite` oder `etf:fornonsmokers`, wodurch es in diesem Zusammenhang zur Einschränkung auf die Domäne „Zimmer/Raum“ dient. So könnte durch das Tripel

unklare Semantik

```
1 <poiUri> etf:fornonsmokers "1"
```

unter der Annahme, das `etf:Rooms` die Bedeutung Raumanzahl hat, darauf geschlossen werden, dass es einen Raum für Nichtraucher gibt. Zieht man aber die Bedeutung der Domäne „Zimmer/Raum“ zur Schlussfolgerung heran und interpretiert für die „1“ den booleschen Wahrheitswert `TRUE`, würde es heißen, dass in allen Räumen dieses POIs ein Rauchverbot herrscht.

Inkonsistenzen

Die Analyse hat ebenfalls einige Inkonsistenzen aufgedeckt. Im Allgemeinen beruht das alte Schema auf dem Prinzip, dass sämtliche Merkmale durch einen booleschen Wert an einen POI gebunden werden. Die „1“ steht dabei für `TRUE`, die „0“ für `FALSE`. Das Tripel

```
1 <poiUri> etf:Dresscode "1"
```

besagt demnach, dass dieser POI eine Kleidungsvorschrift hat. Es gibt aber auch Ausnahmen, was zu Problemen führen kann. So werden

manche Merkmale dazu genutzt, um Mengenangaben zu machen. Dazu soll an dieser Stelle das Beispiel aus dem Listing 5 dienen. Das Tripel aus der ersten Zeile bedeutet unter Annahme des Prinzips der booleschen Werte, dass Betten vorhanden sind. In der zweiten Zeile wird nun aber mit dem selben Merkmal eine Menge von 40 Betten angegeben. Nimmt man jetzt an, das `etf:Beds` immer zur Angabe von Mengen verwendet wird, steht nun das Tripel aus der ersten Zeile für die Aussage, dass nur ein Bett vorhanden ist. Was einen semantischen Unterschied zur der Aussage „es sind Betten vorhanden“ darstellt.

```
1 <poi123> etf:Beds "1" .
2 <poi456> etf:Beds "40" .
```

Listing 5: Beispiel für eine inkonsistente Modellierung innerhalb des alten Merkmals-Schemas.

Abschließend soll auf eine weitere Problematik hingewiesen werden. Ein Merkmal stellt laut dem Schema eine Relation dar. In RDF ist das dadurch gekennzeichnet, dass alle Merkmale vom Typ `rdf:Property` sind. Einige der Merkmale sind aber der Sicht der Ontologie-Entwicklung fehlerhaft modelliert. Ein Beispiel für eine falsche Modellierung stellt das Merkmal für die Angabe der Zugehörigkeit eines POIs zu einer Hotelkette dar. Diese Angabe ist bisher durch ein spezielles Merkmal für jede Hotelkette realisiert. So entspricht das Tripel

fehlerhafte Modellierung

```
1 <poiUri> etf:MercureHotel "1"
```

der Aussage, dass der POI zur Hotelkette „MercureHotel“ gehört. Bei einer Hotelkette handelt es aber um ein Objekt aus der realen Welt und nicht um eine Relation. In RDF sollten deshalb solche Reale-Welt-Objekte auch als eigenständige Ressourcen modelliert werden.

Neben diesen genannten Problemen soll zum Abschluss der Analyse noch kurz auf die Definition eines Merkmals eingegangen werden. Neben der Typisierung als Instanz der Klasse `owl:DatatypeProperty` und der Einordnung in die Hierarchie mittels `rdfs:subPropertyOf`, besitzt ein Merkmal weitere optionale Angaben, welche in der Tabelle 5 aufgelistet sind.

6.1.1 Einfache und Komplexe Werte

Die Analyse der alten Merkmals-Hierarchie hat ergeben, dass touristische Merkmale verschiedene Arten von Werten binden. Dazu gehören Zahlenwerte wie zum Beispiel Mengen-, Größen- und Flächenangaben, Wahrheitswerte, aber auch Objekte aus der realen Welt (Reale-Welt-Objekt). Die folgenden beiden Begriffe sollen zur genauen Unterscheidung der möglichen Arten von Werten dienen, welche durch Merkmale gebunden werden können.

Prädikat	Bedeutung
`vakp`[1]`:hasIcon`	Dient zur Angabe eines kleinen Bildchens.
`rdfs:label`	Bezeichnung des Merkmals. Meist in mehreren Sprachen verfügbar.
`rdfs:comment`	Beinhaltet die Service ID aus der alten relationalen Datenbank.
`dc:description`	Wird zur Angabe einer näheren Erläuterung/Erklärung des Merkmals verwendet.

Tabelle 5: Mögliche Angaben zu einem Merkmal aus dem alten Schema.

Einfacher Wert

Als Einfacher Wert wird in Bezug auf RDF ein Literal bezeichnet. Zusätzlich muss es sich um ein getyptes Literal handeln, um jederzeit die korrekte Semantik des Wertes zu gewährleisten.

Komplexer Wert

Ein Komplexer Wert soll eine RDF-Ressource bezeichnen, welche Instanz einer Klasse ist. In Bezug auf OWL soll es sich um Individuen handeln, die durch eine URI referenziert werden.

6.2 MERKMALE MIT EINFACHEN WERTEN

Zu diesem Muster zählen alle Merkmale, welche getypte Literale, also Zahlenwerte, Zeichenketten und Wahrheitswerte, binden. In der Tourismusdomäne können das Angaben zu Mengen, Größen und Flächen sein, sowie Angaben zu Verboten beziehungsweise Berechtigungen, ausgedrückt durch einen booleschen Wert. Die Modellierung von Merkmalen mit Einfachen Werten ist in `RDF` ohne zusätzliche Definitionen umsetzbar. Für den Einfachen Wert wird ein getyptes Literal erzeugt, welches durch das Merkmal an den POI gebunden wird. Es muss aber beachtet werden, dass die Definition des Einfachen Wertes keine Bindung von Zeichenketten mit einer Sprachangabe durch Merkmale zulässt, da es sich dabei um ungetypte Literale handelt. Von daher sind Angaben, wie Beschreibungen, Bezeichnungen und andere sprachliche Ausdrücke keine touristischen Merkmale im Sinne der Definition dieser Arbeit.

6.2.1 Entwurfsmuster

ALLGEMEINE BESCHREIBUNG

Name	Touristische Merkmale mit Einfachen Werten
Problem	Abbildung eines touristischen Merkmals in RDF, welches Einfache Werte bindet.

1 vakp: <http://vakantieland.nl/properties/>

GRAFISCHE DARSTELLUNG

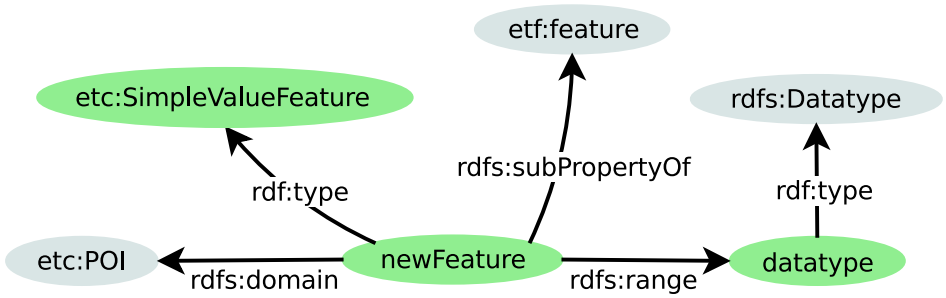


Abbildung 12: Diagramm für die Modellierung von touristischen Merkmalen mit Einfachen Werten in RDF.

ELEMENTE

ETC:SIMPLEVALUEFEATURE Die Klasse aller Merkmale, welche Einfache Werte binden. Ist Subklasse von `owl:DatatypeProperty`, wodurch alle so getypten Merkmale nur Literale mit POIs verbinden können.

NEWFEATURE Das neue Prädikat mit dem Namensraum `etf`, welches das zu modellierende Merkmal repräsentiert. Durch die `rdfs:subPropertyOf`-Relation zu `etf:feature` ist der Definitionsbereich bereits auf die Klasse `etc:POI` festgelegt. Er kann aber zusätzlich durch `rdfs:domain` auf eine spezielle Subklasse von `etc:POI` eingeschränkt werden. Der Wertebereich muss durch `rdfs:range` auf den entsprechenden Datentyp des Einfachen Wertes eingeschränkt werden.

DATATYPE Die Klasse des Datentyps, welcher durch das Merkmal mit einem POI verbunden wird. Der Datentyp muss vom Typ `rdfs:Datatype` sein.

ERSTELLUNGSPROZESS

Folgende Schritte sind notwendig, um ein neues Merkmal, welches Einfache Werte bindet, zu erzeugen:

1. Erzeuge eine neues Prädikat mit dem Namensraum `etf`
2. Definiere das Prädikat als Typ der Klasse `etc:SimpleValueFeature` mittels `rdf:type`
3. Gib eine Bezeichnung mittels `rdfs:label` an
4. Lege die `rdfs:subPropertyOf`-Relation zwischen dem neuen Prädikat und dem Prädikat `etf:feature` fest
5. Wähle einen passenden Datentyp für die möglichen Einfachen Werte

6. Lege den Wertebereich des Prädikats auf den gewählten Datentyp mittels `rdfs:range` fest

7. Soll oder darf das Prädikat nur durch einen bestimmten Typ von POIs verwendet werden? Wenn ja, wähle die entsprechende Subklasse von `etc:POI` aus, sonst überspringe den letzten Schritt

8. Lege den Definitionsbereich des Merkmals auf die gewählte POI-Klasse mittels `rdfs:domain` fest

BEISPIEL

Für ein POI soll angegeben werden können, wie viele Räume dieser zur Verfügung hat. Für die Angabe der Anzahl sind nur ganze positive Zahlenwerte zugelassen. Es ergibt sich daraus die folgende Repräsentation in RDF:

```
etf:numberOfRooms rdf:type etc:SimpleValueFeature ;
                  rdfs:label "Number of rooms"@en ;
                  rdfs:range xsd:nonNegativeInteger ;
                  rdfs:subPropertyOf etf:feature .
```

Listing 6: RDF-Tripel, welche auf der Schema-Ebene für eine Modellierung des Merkmals „Raumanzahl“ erzeugt werden müssen.

Um für einen POI die Raumanzahl anzugeben, muss auf Instanz-Ebene das folgende Tripel erzeugt werden:

```
<poiUri> etf:numberOfRooms "23"^^xsd:nonNegativeInteger .
```

Listing 7: Angabe der Raumanzahl für einen POI in RDF.

6.3 MERKMALE MIT KOMPLEXEN WERTEN

Die Bedeutung des Begriffs Komplexer Wert, unter der Einbeziehung von RDF, wurde bereits in den Vorüberlegungen definiert. Lässt man nun aber für eine Erklärung der Bedeutung den Bezug zu RDF unberücksichtigt, so bezeichnet ein Komplexer Wert eine Reale-Welt-Ressource. Eine Reale-Welt-Ressource kann sowohl ein anfassbares, konkretes Objekt, aber auch ein imaginäres, abstraktes Objekt, wie zum Beispiel ein Konzept oder eine Institution, sein.

6.3.1 Konkrete vs. abstrakte Komplexe Werte

Die touristischen Komplexen Werte lassen sich in zwei unterschiedliche Gruppen aufteilen. Zur Erläuterung soll das folgende Beispiel zweier zufälliger POIs, welche vom Typ *Hotel* sind, dienen. Beide Hotels gehören zudem der gleichen Hotelkette an. In RDF ausgedrückt, existieren also

zwei Tripel, die jeweils das Subjekt, eines der beiden Hotels, durch das touristische Merkmal `gehörtZu` mit ein und dem selben Komplexen Wert, der Hotelkette, verbinden. Die Semantik ist an dieser Stelle eindeutig. Die Hotelkette stellt somit einen konkreten Komplexen Wert dar. Betrachten wir aber nun den folgenden Fall. Beide Hotels sollen zusätzlich einen Aufzug besitzen. Würde man wie im Fall zuvor, bei der Modellierung in `RDF` für den Aufzug ein und den selben Komplexen Wert verwenden, so wäre die Semantik an dieser Stelle nicht mehr eindeutig. Es könnte darauf geschlossen werden, dass beide Hotels sich ein und den selben Aufzug teilen. Häufig soll es sich bei einer solchen Angabe aber eher um die Aussage handeln, dass ein Aufzug vorhanden ist. Es wird somit nicht ein bestimmtes anfassbares Reale-Welt-Objekt benannt, sondern die Abstraktion dieses Objekts. In diesem Fall kann also von einem Konzept, einem abstrakten Komplexen Wert, gesprochen werden, für den die Semantik zusätzlich geklärt werden muss, um Fehlinterpretationen zu vermeiden. Im Folgenden soll deswegen erläutert werden, wie konkrete und abstrakte Komplexe Werte in `RDF` modelliert werden können, damit die Semantik eindeutig bleibt.

konkreter Komplexer Wert

Ein konkreter Komplexer Wert existiert nur einmal auf der Welt. Es handelt sich demnach um ein eindeutig bestimmbares Reale-Welt-Objekt mit einem eindeutigen Identifikator. Die Modellierung solcher Ressourcen in `RDF` ist trivial. Für das Objekt wird eine URI definiert. Zusätzlich können weitere Aussagen über diese Ressource getroffen werden, ohne dass Beziehungen, an denen die Ressource an der Stelle des Objekts steht, beeinflusst werden. Des Weiteren muss die Semantik für konkrete Komplexe Werte nicht weiter spezifiziert werden.

abstrakter Komplexer Wert

Dagegen beschreibt ein abstrakter Komplexer Wert die Abstraktion eines nicht näher konkretisierbaren Reale-Welt-Objekts. Mit Hilfe eines abstrakten Komplexen Wertes soll kein konkretes Reale-Welt-Objekt beschrieben werden, sondern die Abstraktion dessen. Um Beziehungen zu solchen Abstraktionen darzustellen, besteht in `RDF` die Möglichkeit Klassen an der Objektposition in einem Tripel zu verwenden. Verwendet man aber Klassen als Werte für Prädikate, kann der Wertebereich (`rdfs:range`) nur auf `owl:Class` eingeschränkt werden. Dies würde bedeuten, dass jede beliebige OWL-Klasse als Wert eingesetzt werden kann, womit der Wertebereich letztendlich wieder als uneingeschränkt zu betrachten ist. Laut der Anforderung *A010* muss aber eine Einschränkung des Wertebereichs möglich sein. Um zu vermeiden, dass Klassen als Werte für Prädikate genutzt werden, könnte man auch Instanzen zur Repräsentation abstrakter Komplexer Werte verwenden. So würde man zum Beispiel für die Klasse `Sanitäreinrichtung` die Instanzen `Dusche`, `Badewanne`, `Toilette` und so weiter erzeugen. Durch diese Variante wäre es möglich, den Wertebereich auf eine konkrete Klasse festzulegen, sodass nur Instanzen dieser Klasse als Werte für das entsprechende Prädikat zulässig sind. Dadurch wäre aber, wie bereits zu Anfang des Abschnitts erwähnt,

die Semantik solcher abstrakten Komplexen Werte nicht mehr eindeutig. Durch eine Einführung einer gesonderten Semantik für abstrakte Komplexe Werte könnte dieses Problem gelöst werden. Da es sich bei einem abstrakten Komplexen Wert um eine Abstraktion eines konkreten Reale-Welt-Objekts handelt, kann man an dieser Stelle auch von einem Konzept sprechen. Ein abstrakter Komplexer Wert beschreibt also das Konzept eines Reale-Welt-Objekts. Mit Hilfe des SKOS-Vokabulars [Miles u. Bechhofer, 2008] besteht die Möglichkeit, solche Konzepte in RDF zu modellieren. Das zentrale Element dieses Vokabulars stellt `skos:Concept` dar, die Klasse aller Konzepte. Der einfachste Weg besteht nun darin, für jeden abstrakten Komplexen Wert, zum Beispiel `Dusche`, eine Instanz vom Typ `skos:Concept` anzulegen. Dies bedeutet aber wiederum, dass für alle Prädikate, welche solche Instanzen als Werte binden, der Wertebereich nur auf die Klasse `skos:Concept` festgelegt werden kann. Es soll aber wie schon gesagt möglich sein, für jedes Prädikat den Wertebereich unabhängig zu definieren. Um diesen nun doch getrennt voneinander festlegen zu können, besteht die Möglichkeit SKOS und OWL parallel anzuwenden [Miles, 2007]. Das heißt, für jedes Hauptkonzept, wie *Sanitäreinrichtung*, wird eine Klasse erzeugt, welche zusätzlich eine Subklasse von `skos:Concept` ist. Somit kann diese Klasse als Wertebereich für bestimmte Prädikate verwendet werden und gleichzeitig handelt es sich um eine Konzeptklasse, wodurch alle Instanzen, beispielweise `Dusche`, Konzepte darstellen. Ein weiterer Vorteil besteht darin, dass diese Konzepte zusätzlich durch die SKOS-Elemente `skos:broader` (Oberbegriff) und `skos:narrower` (Unterbegriff) in eine Hierarchie gebracht werden können. Beispielsweise ist das Konzept *Warmdusche* eine spezialisiertere Beschreibung (`skos:narrower`) des Konzepts *Dusche*.

Zusammenfassung

Wie in diesem Abschnitt gezeigt, gibt es unterschiedliche Gruppen von Komplexen Werten in der Tourismusdomäne. Konkrete Komplexe Werte lassen sich mit Hilfe von RDF nativ modellieren. Bei abstrakten Komplexen Werten ist die Semantik problematisch, welche aber durch die Verwendung des SKOS-Vokabulars geklärt werden kann. Wie bei den konkreten Komplexen Werten auch, werden Klassen definiert, welche jedoch zusätzlich Subklasse von `skos:Concept` sind, wodurch alle Instanzen dieser Klasse Konzepte darstellen und damit die Semantik eindeutig festgelegt ist.

6.3.2 Entwurfsmuster

ALLGEMEINE BESCHREIBUNG

Name	Touristische Merkmale mit Komplexen Werten
Problem	Abbildung eines touristischen Merkmals in RDF, welches Komplexe Werte bindet. Um dies semantisch eindeutig und korrekt abbilden zu können, muss zwischen abstrakten und konkreten Komplexen Werten unterschieden werden.

GRAFISCHE DARSTELLUNG

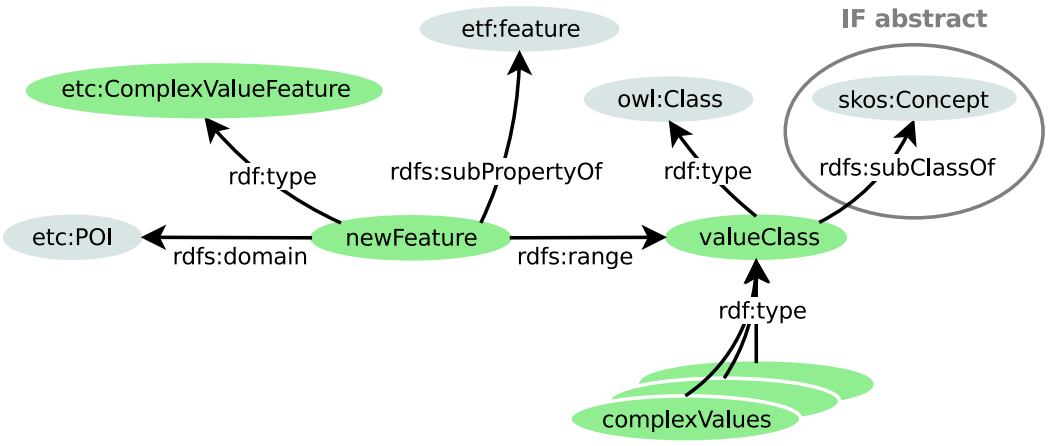


Abbildung 13: Muster für die Modellierung von touristischen Merkmalen mit Komplexen Werten in RDF.

ELEMENTE

ETC:COMPLEXVALUEFEATURE Die Klasse aller Merkmale, welche Komplexe Werte binden. Ist Subklasse von `owl:ObjectProperty`, wodurch alle so getypten Merkmale nur Individuen mit POIs verbinden können.

NEWFEATURE Das neue Prädikat mit dem Namensraum `etf`, welches das zu modellierende Merkmal repräsentiert. Durch die `rdfs:subPropertyOf`-Relation zu `etf:feature` ist der Definitionsbereich bereits auf die Klasse `etc:POI` festgelegt. Er kann aber zusätzlich durch `rdfs:domain` auf eine spezielle Subklasse von `etc:POI` eingeschränkt werden.

VALUECLASS Die Klasse der Individuen mit Namensraum `etc`, welche durch das Merkmal mit einem POI verbunden werden. Handelt es sich bei den Instanzen um abstrakte Komplexe Werte, so muss diese Klasse auch als Subklasse von `skos:Concept` deklariert werden.

COMPLEXVALUES Die Menge aller Komplexen Werte. Diese müssen vom Typ der definierten Klasse `valueClass` sein.

ERSTELLUNGSPROZESS

Folgende Schritte sind notwendig, um ein neues Merkmal, welches Komplexe Werte bindet, zu erzeugen:

1. Erzeuge eine neues Prädikat mit dem Namensraum `etf`
2. Definiere das Prädikat als Typ der Klasse `etc:ComplexValueFeature` mittels `rdf:type`
3. Gib eine Bezeichnung mittels `rdfs:label` an
4. Lege die `rdfs:subPropertyOf`-Relation zwischen dem neuen Prädikat und dem Prädikat `etf:feature` fest
5. Erzeuge eine neue Klasse für die Komplexen Werte mit dem Namensraum `etc`
6. Definiere die Klasse als `owl:Class` mittels `rdf:type`
7. Falls es sich bei den Instanzen um abstrakte Komplexe Werte handelt, so definiere die Klasse zusätzlich als Subklasse von `skos:Concepte` mittels `rdfs:subClassOf`
8. Lege den Wertebereich des Prädikats auf die neu definierte Klasse mittels rdfs:range fest
9. Soll oder darf das Prädikat nur durch einen bestimmten Typ von POIs verwendet werden? Wenn ja, wähle die entsprechende Subklasse von `etc:POI` aus, sonst überspringe den nächsten Schritt
10. Lege den Definitionsbereich des Merkmals auf die gewählte POI-Klasse mittels `rdfs:domain` fest
11. Definiere die Menge der Komplexen Werte, für jedes Element dieser Menge:
    a) Erzeuge auf Instanz-Ebene eine neue Ressource der unter *5.* definierten Klasse mittels `rdf:type`
    b) Gib eine Bezeichnung mittels `rdfs:label` an

BEISPIEL

Merkmal mit konkreten Komplexen Werten Für Hotels soll die Zugehörigkeit zu einer Hotelkette angegeben werden. Dabei können mehrere Hotels der selben Hotelkette angehören. Es ergibt sich daraus die folgende Modellierung in RDF:

```
etf:affiliation rdf:type etc:ComplexValueFeature ;
                rdfs:label "Affiliation"@en ;
                rdfs:range etc:HotelChain ;
                rdfs:subPropertyOf etf:feature .

etc:HotelChain  rdf:type owl:Class ;
                rdfs:label "Hotel chain"@en .
```

Listing 8: RDF-Tripel, welche auf der Schema-Ebene für eine Modellierung des Merkmals „Zugehörigkeit“ erzeugt werden müssen.

Auf Instanz-Ebene muss dann die entsprechende Hotelkette definiert und an einem POI wie folgt angegeben werden:

```
@prefix eto:<http://vakantieland.nl/data/tourismObjects/> .
eto:MercureHotel rdf:type etc:HotelChain ;
                 rdfs:label "Mercure Hotel"@en .
<poiUri> etf:affiliation eto:MercureHotel .
```

Listing 9: Modellierung der Zugehörigkeit eines POI zu einer Hotelkette mittels RDF.

Merkmal mit abstraken Komplexen Werten Es soll möglich sein, für POIs Ausstattungsmerkmale anzugeben. Bei Ausstattungsmerkmalen handelt es sich im Allgemeinen um Konzepte wie eine Dusche, eine Terrasse oder eine Minibar. Daraus ergibt sich die folgende Modellierung in RDF:

```
etf:hasAmenities rdf:type etc:ComplexValueFeature ;
                 rdfs:label "Amenities"@en ;
                 rdfs:range etc:Equipment ;
                 rdfs:subPropertyOf etf:feature .

etc:Equipment    rdf:type owl:Class ;
                 rdfs:subClassOf skos:Concept ;
                 rdfs:label "Equipment"@en .
```

Listing 10: RDF-Tripel, welche auf der Schema-Ebene für eine Modellierung des Merkmals „Ausstattung“ erzeugt werden müssen.

Um für einen POI ein Ausstattungsmerkmal auf Instanz-Ebene anzugeben, muss dieses definiert und wie folgt angeben werden:

```
@prefix eto:<http://vakantieland.nl/data/tourismObjects/> .
eto:Shower rdf:type etc:Equipment ;
           rdfs:label "Shower"@en .
<poiUri> etf:hasAmenities eto:Shower .
```

Listing 11: Angabe einer Dusche als Ausstattungsmerkmal für einen POI in RDF.

6.4 POLYVALENTE MERKMALE

Der Begriff „polyvalent“ ist zusammengesetzt aus dem griechischem Wort „poly“, welches übersetzt für „viel“ steht, und dem lateinischen Wort „valēns“, welches übersetzt „wert“ heißt. Beide Wörter zusammen stehen also für die Bedeutung „vielwertig“ oder auch „mehrwertig“. Merkmale sind normalerweise binär, das heißt sie verbinden einen POI mit einem Wert. Jedoch gibt es Sonderfälle, in denen ein Merkmal mehr als einen Wert gleichzeitig mit einem POI verbinden soll. Diese sollen in dieser Arbeit als Polyvalente Merkmale bezeichnet werden.

Als Beispiel für ein Polyvalentes Merkmal soll das Merkmal `ex:inDerNaeheVon` dienen. Mit Hilfe dieses Merkmals kann für ein POI angegeben werden, dass dieser in der Nähe von etwas liegt, beispielsweise in der Nähe eines Sees. Diese Angabe allein ist aber für einen Touristen wenig hilfreich. So könnte der See etwa 500 Meter entfernt sein, wodurch er problemlos zu Fuß erreichbar wäre. Er könnte aber auch 20 Kilometer weit weg sein, wofür dann schon ein Fahrzeug gebraucht wird, um den See in annehmbarer Zeit zu erreichen. Damit die Information, dass etwas in der Nähe liegt, für den Touristen auch von Nutzen ist, muss somit zusätzlich noch die Entfernung angegeben werden können.

Polyvalente Beziehungen können in `RDF` auf verschiedenen Wegen abgebildet werden. Eine Variante ist die Verwendung des `RDF Reification`-Vokabulars [Manola u. Miller, 2004]. Es bietet die Möglichkeit, Aussagen über ein bestimmtes Tripel zu treffen („eine Aussage über eine Aussage“). Das reifizierte Tripel kann also an der Subjekt- oder Objekt-Position in einem weiteren Tripel verwendet werden. Dazu beinhaltet das Vokabular die folgenden Elemente: den Typ `rdf:Statement` und die Prädikate `rdf:subject`, `rdf:predicate` und `rdf:object`. Eine Reifikation des obigen Beispiels zeigt das Listing 12:

```
1 ex:poi123       ex:inDerNaeheVon          ex:See .
2 ex:triple12345  rdf:type                  rdf:Statement ;
3                 rdf:subject               ex:poi123 ;
4                 rdf:predicate             ex:inDerNaeheVon ;
5                 rdf:object                ex:See ;
6                 ex:EntfernungInKilometer  "20"^^xsd:decimal .
```

Listing 12: Beispiel für eine Reifikation in RDF.

Wie im Listing 12 zu sehen, sind bei der Anwendung von Reifikation vier Tripel involviert, welche auch als “reification quad“ bezeichnet werden. Die Tatsache, dass immer vier Tripel definiert werden müssen, damit ein bestimmtes Tripel reifiziert werden kann, erzeugt zusätzlichen Aufwand bei der Erstellung und Auswertung der in dieser Form gespeicherten Informationen, was der Anforderung *A040* widerspricht. Ein weiterer Nachteil besteht darin, dass die Semantik bei der Verwendung des reifizierten

Tripels als Subjekt bzw. Objekt in einem anderen Tripel nicht eindeutig ist. So ist bei dem Tripel

```
ex:triple12345 ex:EntfernungInKilometer "20"^^xsd:decimal
```

aus dem Listing 12 nicht klar, auf wen oder was sich die Entfernung bezieht. Es müsste für das Prädikat `ex:EntfernungInKilometer` gesondert festgelegt werden, dass sich diese Entfernung auf den Abstand zwischen Subjekt und Objekt aus dem reifizierten Tripel bezieht. Diese und weitere Probleme sind auch der Grund dafür, warum das `RDF Reification`-Vokabular im Allgemeinen nur wenig eingesetzt wird. Somit kommt es für die Modellierung von Polyvalenten Merkmalen nicht in Frage. Eine Alternative zur Reifikation stellen die sogenannten `N-ary Relations` [Noy u. Rector, 2006] dar. Sie ermöglichen es, eine Ressource mit mehr als einer Ressource oder einem Wert zu verbinden. Das Grundprinzip sieht vor, das die Relation durch eine Klasse repräsentiert wird. Eine Instanz dieser Klasse stellt eine Instanz der Relation dar. Die Werte werden nun wiederum durch Prädikate an diese Instanz gebunden. Zur Veranschaulichung zeigt das Listing 13, wie man das bisher verwendete Beispiel mit Hilfe der `N-ary Relations` modellieren würde.

```
ex:poi123  ex:inDerNaeheVon           _:KnotenX .
_:KnotenX  rdf:type                   ex:InDerNaeheVon_Relation ;
           ex:NaheliegendesObjekt     ex:See ;
           ex:EntfernungInKilometer   "20"^^xsd:decimal .
```

Listing 13: Beispiel für eine `N-ary Relations`-Modellierung.

Auch bei dieser Variante müssen, bevor die eigentlichen Informationen an das Subjekt gebunden werden können, zwei zusätzliche Tripel erzeugt werden. Des Weiteren muss die Semantik der Prädikate, mit denen die Ressourcen beziehungsweise Werte an die Instanz der Relation geknüpft werden, gesondert festgelegt werden. So ist das eigentliche Subjekt nicht die Relation selbst, sondern die Ressource, welche als Subjekt mit der Instanz der Relation verknüpft ist. Insgesamt betrachtet, würde sich ein Einsatz von `N-ary Relations` für die Modellierung von Polyvalenten Merkmalen eignen. Die Analyse des alten Merkmals-Schemas hat aber gezeigt, dass in allen Fällen, welche auf dieses Muster zutreffen, nur eine Ressource und ein weiterer Einfacher Wert gleichzeitig mit einem POI verknüpft werden müssen. Es handelt sich hier also um eine dreistellige Beziehung. Diese Tatsache führt zu einem weiteren Lösungsansatz. Dieser besteht darin, die unterschiedlichen Ausprägungen des Einfachen Wertes direkt über das Merkmal selbst abzubilden. Für jedes Datum oder auch Bereiche, den der Einfache Wert einnehmen kann, wird ein Submerkmal erzeugt. Das Listing 14 veranschaulicht, wie dies für das in diesem Abschnitt genutzte Beispiel aussehen könnte.

```
ex:inDerNaeheVon20 rdfs:subPropertyOf ex:inDerNaeheVon .
ex:poi123          ex:inDerNaeheVon20 ex:See .
```

Listing 14: Beispielhafte Modellierung für ein Polyvalentes Merkmal in RDF.

Damit die Semantik in diesem Fall eindeutig ist, muss an dem Merkmal die Interpretation (meist die Einheit) des Einfachen Wertes definiert werden. Für `ex:inDerNaeheVon` sollte zum Beispiel mit Hilfe eines Labels (`rdfs:label`) oder einer Beschreibung (`dc:description`) festgelegt werden, dass alle Angaben durch die Submerkmale in Kilometern zu interpretieren sind. Um die Menge von Submerkmalen überschaubar zu halten, bietet es sich zudem an, insofern möglich, das ein Submerkmal einen bestimmten Bereich der potenziellen Werte des Einfachen Wertes abdeckt. So könnte `ex:inDerNaeheVon0.5` eine Entfernung bis zu, anstatt genau, 500 Meter bedeuten. Ein weiterer Vorteil dieses Ansatzes ist die Kompatibilität zu den bisher definierten Mustern 6.2 und 6.3. Durch die Modellierung der Polyvalenten Merkmale unter Verwendung einer Menge von Submerkmalen, ist es weiterhin möglich, das Merkmal selbst als `etc:SimpleValueFeature` oder `etc:ComplexValueFeature` auszuzeichnen. Das heißt, für das Merkmal `ex:inDerNaeheVon` kann zusätzlich definiert werden, ob es Einfache oder Komplexe Werte bindet.

Zusammenfassung

Wie gezeigt, können Polyvalente Merkmale durch die Verwendung einer Menge von Submerkmalen abgebildet werden. Dabei repräsentieren die Submerkmale die verschiedenen Ausprägungen eines in die mehrstellige Beziehung involvierten Einfachen Wertes. Durch diesen Ansatz ist zudem nur ein einziges Tripel zur Abbildung des Polyvalentes Merkmals notwendig. Außerdem kann diese Lösung mit den bisher definierten Entwurfsmustern zur Modellierung von Merkmalen kombiniert werden.

6.4.1 Entwurfsmuster

ALLGEMEINE BESCHREIBUNG

Name	Polyvalente touristische Merkmale
Problem	Abbildung eines touristischen Merkmals in RDF, welches zwei Werte gleichzeitig bindet. Bei den Werten kann es sich entweder um einen Einfachen und einen Komplexen Wert oder um zwei Einfache Werte handeln.

GRAFISCHE DARSTELLUNG

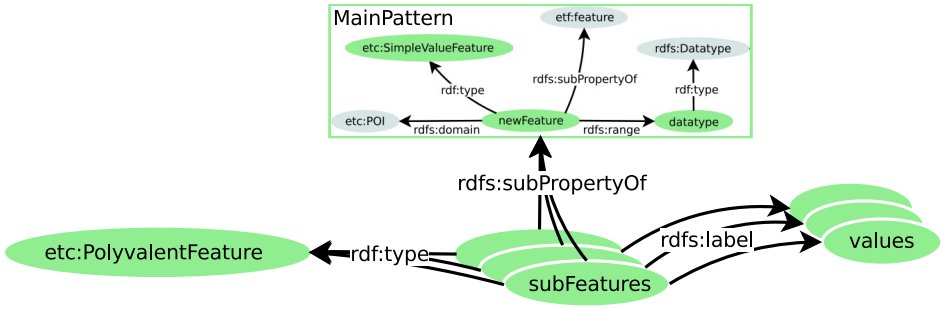


Abbildung 14: Muster für die Modellierung von Polyvalenten Merkmalen in RDF.

ELEMENTE

ETC:POLYVALENTFEATURE Die Klasse aller Polyvalenten Merkmale. Ist Subklasse von `rdf:Property`. Polyvalente Merkmale haben somit vorerst keine Einschränkung bezügliches ihres Wertebereichs. Jedes Polyvalente Merkmal ist Submerkmal eines bestimmten Merkmals, welches durch ein anderes Entwurfsmuster (`MainPattern`) definiert wird.

MAINPATTERN Die Modellierung eines Polyvalenten Merkmals wird durch die Erweiterung eines anderen Entwurfsmusters realisiert. Aktuell werden die Entwurfsmuster für Merkmale mit Einfachen und Komplexen Werten unterstützt. Das Merkmal, welches um Polyvalente Merkmale erweitert werden soll, wird zunächst nach dem ausgewählten Entwurfsmuster modelliert.

SUBFEATURES Die Menge der Polyvalenten Merkmale mit Namensraum `etf`, welche alle Subprädikat vom entsprechenden Merkmals-Prädikat aus dem ausgewählten Entwurfsmuster `MainPattern` und Typ der Klasse `etc:PolyvalentFeature` sind. Durch diese Relation wird der Wertebereich der Polyvalenten Merkmale auf den im Hauptentwurfsmuster festgelegten Wertebereich eingeschränkt. Die URI eines Polyvalenten Merkmals setzt sich aus der URI des Supermerkmals plus der Ausprägung des jeweiligen Einfachen Wertes aus `values` zusammen, den das Polyvalente Merkmal abbilden soll. Als Bezeichnung für ein Polyvalentes Merkmal wird ebenfalls die Ausprägung des jeweiligen Einfachen Wertes aus `values` verwendet.

VALUES Die Menge der Ausprägungen des Einfachen Wertes, welche durch die Polyvalenten Merkmale repräsentiert werden sollen.

HINWEISE

In manchen Fällen ist es nicht sinnvoll, jede mögliche Ausprägung eines Einfachen Wertes durch ein Polyvalentes Merkmal abzubilden. Insofern es die Semantik zulässt, sollten Bereiche definiert werden, die dann jeweils durch ein Polyvalentes Merkmal repräsentiert werden (siehe Beispiel aus der Diskussion in Abschnitt 6.4 auf der Seite 46).

Falls durch die Polyvalenten Merkmale zwei Einfache Werte gebunden werden, sollte derjenige Einfache Wert zur Erzeugung der Polyvalenten Merkmale ausgewählt werden, der eine geringere Zahl an möglichen Ausprägungen aufweißt. Dadurch kann die Anzahl der zu erzeugenden Polyvalenten Merkmale so gering wie möglich gehalten werden.

ERSTELLUNGSPROZESS

Folgende Schritte sind notwendig, um Polyvalente Merkmale zu erzeugen:

1. Wähle ein Entwurfsmuster für das Merkmal, welches um Polyvalente Merkmale erweitert werden soll
2. Führe die Schritte des Erstellungsprozess des gewählten Entwurfsmusters durch
3. Lege die durch Polyvalente Merkmale abzubildenden Ausprägungen des Einfachen Wertes fest, wenn möglich, sollten Bereiche festgelegt werden, um die Menge der Polyvalenten Merkmale klein zu halten
4. Definiere die Menge der zu erzeugenden Polyvalenten Merkmale, für jedes Element dieser Menge:
   a) Erzeuge ein neues Prädikat mit der URI: URI des Supermerkmals + Ausprägung des Einfachen Wertes
   b) Definiere das Prädikat als Typ der Klasse `etc:PolyvalentFeature`
   c) Gib eine Bezeichnung mittels `rdfs:label` an, welche die jeweilige Ausprägung des Einfachen Wertes enthält
   d) Lege die `rdfs:subPropertyOf`-Relation zwischen dem Polyvalenten Merkmal und dem Supermerkmal fest

BEISPIEL

POIs können durch Organisationen verschiedene Auszeichnungen erhalten. Diese Auszeichnungen werden jährlich verliehen. Die Problemstellung kann mit dem Muster für Polyvalente Merkmale wie folgt in RDF modelliert werden:

```
etf:acknowledge       rdf:type etc:ComplexValueFeature ;
                      rdfs:label "Acknowledge"@en ;
                      rdfs:range etc:Acknowledgment ;
                      rdfs:subPropertyOf etf:feature .

...

etf:acknowledge2004 rdf:type etc:PolyvalentFeature ;
                      rdfs:label "2004"^^xsd:integer ;
                      rdfs:subPropertyOf etf:acknowledge .

etf:acknowledge2005 rdf:type etc:PolyvalentFeature ;
                      rdfs:label "2005"^^xsd:integer ;
                      rdfs:subPropertyOf etf:acknowledge .

...

etc:Acknowledgment  rdf:type owl:Class ;
                      rdfs:label "Acknowledgment"@en .
```

Listing 15: RDF-Tripel, welche auf der Schema-Ebene für eine Modellierung des Merkmals „Auszeichnung“ erzeugt werden müssen. Für das Hauptmerkmal wurde das Entwurfsmuster für Konkrete Komplexe Werte angewandt.

Für einen POI soll nun eine Auszeichnung aus dem Jahre 2004 angegeben werden. Dazu müssen auf Instanz-Ebene die folgenden Tripel erzeugt werden:

```
@prefix eto:<http://vakantieland.nl/data/tourismObjects/> .
eto:ADACrecommended rdf:type etc:Acknowledgment ;
                      rdfs:label "ADAC recommended"@en .
<poiUri> etf:acknowledge2004 eto:ADACrecommended .
```

Listing 16: Angabe einer Auszeichnung aus dem Jahre 2004 für einen POI in RDF.

7 EVOLUTION DER MERKMALE

Inhaltsangabe

Im vorhergehenden Kapitel wurden die Ontologie-Entwurfsmuster zur einheitlichen Modellierung von touristischen Merkmalen definiert. In diesem Kapitel werden nun Anleitungen zur Evolution der Merkmale entwickelt, die vorgeben, wie die Merkmale aus dem alten Schema in das neue Schema überführt werden sollen. Zu Beginn dieses Kapitels wird eine Evolutionsstrategie auf Basis der Analyse des alten Schemas und der definierten Entwurfsmuster erstellt. Anschließend wird auf die Änderungsermittlung eingegangen. Hier werden tabellarisch sämtliche Entscheidungen vorgestellt, für welches Merkmal beziehungsweise Domäne von Merkmalen, welches Ontologie-Entwurfsmuster angewandt werden soll. Basierend auf der Evolutionsstrategie und den Erfahrungen aus der Änderungsermittlung werden anschließend die verschiedenen Algorithmen zur Evolution der Merkmale an einem passenden Beispiel erläutert und die Anpassungsschritte abstrakt definiert. Einige Schritte können später bei der Implementierung optional gestaltet werden, um spezielle Szenarien erfüllen zu können. Zur Evaluation der korrekten Evolution eines Merkmals ist es beispielsweise von Vorteil, wenn der Schritt, in dem es zur Löschung der alten Daten aus der Ontologie kommt, ausgelassen werden kann.

7.1 EVOLUTIONSSTRATEGIE

An dieser Stelle soll eine Strategie festgelegt werden, wie die einzelnen Merkmale des alten Schemas jeweils angepasst werden sollen. Da durch die Ontologie-Entwurfsmuster die Kriterien für das Aussehen des neuen Merkmals-Schemas bereits festgelegt sind, handelt es sich hierbei um eine Struktur-getriebene Strategie (siehe Abschnitt 4.1.2). Folgende Strate-

giepunkte ergeben sich aus den bisherigen Erkenntnissen bezüglich der Analyse des alten Schemas:

- Wird ein Merkmal auf Instanz-Ebene dazu verwendet, um Zahlenwerte, die nicht ausschließlich „1“ darstellen, mit POIs zu verknüpfen, handelt es sich also um Einfache Werte, so wird es entsprechend dem Muster „Merkmale mit Einfachen Werten“ angepasst, siehe Abschnitt 7.3.
- Besitzt ein Merkmal Submerkmale und handelt es sich bei denen um Individuen und demnach um Komplexe Werte, so werden das Merkmal und die Submerkmale entsprechend dem Muster „Merkmale mit Komplexen Werten“ angepasst, siehe Abschnitt 7.4.
- Sollen durch ein Merkmal zwei Werte gleichzeitig gebunden werden, so muss eine Anpassung entsprechend dem Entwurfsmuster „Polyvalente Merkmale“ durchgeführt werden, siehe Abschnitt 7.5.
- Trifft keiner der oberen Punkte zu, so muss zwischen den folgenden Unterpunkten entschieden werden:
  - Wird das Merkmal (und die eventuell vorhanden Submerkmale) auf der Instanz-Ebene nicht genutzt, so soll es gelöscht werden, siehe Abschnitt 7.6.
  - Entsprechen zwei oder mehrere Merkmale der selben Semantik, so sollen diese miteinander verschmolzen werden, siehe Abschnitt 7.7
  - Ist das Merkmal in seiner Semantik fragwürdig, handelt es sich zum Beispiel nicht um ein touristisches Merkmal im Sinne der Definition dieser Arbeit, so soll es temporär aus der Hierarchie entfernt und für eine spätere Evolution markiert, jedoch nicht gelöscht werden. Siehe Abschnitt 7.8.
  - Kann wiederum keiner der Punkte angewandt werden, so soll das Merkmal und dessen etwaige Submerkmale unverändert in das neue Schema übernommen werden.

7.2 ÄNDERUNGSERMITTLUNG

Für die einzelnen Entscheidungen, welche Merkmale an welches Ontologie-Entwurfsmuster angepasst werden sollen, wurde eine Mischung aus Struktur- und Daten-getriebener Änderungsermittlung angewandt. Für die Struktur-getriebene Ermittlung wurde eine Tabelle mit allen alten Merkmalen und deren jeweiligen Supermerkmalen, welche die Domäne bestimmen, erstellt. Parallel dazu, als Daten-getriebener Ansatz, wurde zusätzlich noch eine Tabelle mit all denjenigen alten Merkmalen aufbereitet, welche einen Literalwert ungleich „1“ besaßen. Auf

Basis dieser beiden Tabellen und zusammen mit dem Auftraggeber des Vakantieland-Projekts wurde zunächst die Bedeutung jedes einzelnen Merkmals durchgesprochen. Mit Hilfe dieser Analyse konnten dann anschließend die Entscheidungen, welches Merkmal wie angepasst werden soll, getroffen werden. Es würde an dieser Stelle den Rahmen der Arbeit überschreiten, für jede Entscheidung eine Begründung anzugeben. Daher sind in der Tabelle 10, zu finden im Anhang A.1 auf der Seite 92, nur die jeweiligen Entscheidungen ohne eine Begründung angegeben. An dem Merkmal `etf:Keten` und dessen Submerkmalen soll aber beispielhaft dieser Entscheidungsprozess gezeigt werden. Jedes Submerkmal von `etf:Keten` steht für eine bestimmte Hotelkette, so repräsentiert `etf:MercureHotel`, wie die URI bereits vermuten lässt, die Hotelkette „Mercure Hotel“. Laut den Festlegungen muss damit für `etf:Keten` das Muster für „Merkmale mit Komplexen Werten“ angewandt werden. Da mehrere POIs gleichzeitig zu ein und der selben Hotelkette gehören können, müssen die einzelnen Hotelketten als konkrete Komplexe Werte modelliert werden.

7.3 EVOLUTION ZU MERKMAL MIT EINFACHEN WERTEN

Das alte Schema beinhaltet Merkmale, welche auf der Instanz-Ebene zur Verknüpfung unterschiedlicher Zahlenwerte mit POIs genutzt werden. So dient etwa das Merkmal `etf:Rooms` zur Angabe der Raumanzahl eines POIs. Einen Auszug aus diesen Angaben zeigt das Listing 17.

```
1 <http://vakantieland.nl/data/pois/rookshuus>    etf:Rooms "1" .
2 <http://vakantieland.nl/data/pois/camp.venlo>   etf:Rooms "49" .
3 <http://vakantieland.nl/data/pois/hessels>      etf:Rooms "11" .
4 <http://vakantieland.nl/data/pois/ons.centrum> etf:Rooms "74" .
5 ...
```

Listing 17: Ausschnitt aus der Verwendung des Merkmals `etf:Rooms` auf Instanz-Ebene.

Für eine erfolgreiche Evolution zu einem Merkmal mit Einfachen Werten muss zunächst ein neues Merkmal entsprechend dem Entwurfsmuster erzeugt und danach Angaben, wie Bezeichnung, Icon und so weiter, vom alten Merkmal kopiert werden. Da wie im Listing 17 zu sehen, für die Literale kein Datentyp angegeben ist, müssen diese außerdem auf einen korrekten Datentyp festgelegt werden.

ANPASSUNGSSCHRITTE

1. Erzeugen eines neuen Merkmals entsprechend dem Entwurfsmuster
2. Kopieren der Bezeichner (`rdfs:label`)

3. Kopieren der Kommentare (`rdfs:comment`)
4. Kopieren der Beschreibungen (`dc:description`)
5. Kopieren des Icons (`vakp:hasIcon`)
6. Ermittlung aller Tripel aus der Instanz-Ebene, in denen das alte Merkmal verwendet wird, für jedes dieser Tripel unter Beibehaltung des Subjekts (URI des POIs) wird ein neues Tripel erzeugt, durch:
   a) Austausch des alten Merkmals gegen das neue Merkmal
   b) Typisierung der Literalwerte auf den korrekten Datentyp
7. Löschen der alten Instanz-Tripel
8. Löschen des alten Merkmals

7.4 EVOLUTION ZU MERKMAL MIT KOMPLEXEN WERTEN

Viele Merkmale aus dem alten Schema stellen im eigentlichen Sinne keine Relationen, sondern Individuen dar. Zum Beispiel alle Submerkmale von `etf:Rest.keuken`: `etf:American`, `etf:Caribbean` oder auch `etf:Fish`. Das Listing 18 zeigt, wie diese Merkmale auf Instanz-Ebene mit den einzelnen POIs verbunden sind. Es werden dabei nur die Submerkmale genutzt, nicht aber das Merkmal `etf:Rest.keuken` selbst.

```
1 <http://vakantieland.nl/data/pois/alexandria> etf:American "1" .
2 etf:American rdfs:subPropertyOf etf:Rest.keuken .
3 <http://vakantieland.nl/data/pois/casabassin> etf:Caribbean "1" .
4 etf:Caribbean rdfs:subPropertyOf etf:Rest.keuken .
5 <http://vakantieland.nl/data/pois/brittenburg> etf:Fish "1" .
6 etf:Fish rdfs:subPropertyOf etf:Rest.keuken .
7 ...
```

Listing 18: Verwendung der Submerkmale von `etf:Rest.keuken` auf Instanz-Ebene.

Zur Evolution von Merkmalen, welche dem hier gezeigten Beispiel entsprechen, muss ein neues Merkmal entsprechend dem Muster „Merkmale mit Komplexen Werten“ erzeugt und alle Submerkmale von `etf:Rest.keuken` müssen zu Komplexen Werten umgewandelt werden. Dabei sollen sie als Instanzen der Klasse aus dem Entwurfsmuster typisiert werden. Die Literale mit dem Wert „1“, die durch die Submerkmale mit den POIs verbunden werden, geben lediglich an, dass ein POI über jenes Merkmal verfügt und spielen somit für die Evolution keine Rolle. Die Merkmalsangaben für die POIs müssen demnach so angepasst werden, dass die alten Submerkmale jeweils durch das neue Merkmal ersetzt

werden und als RDF-Objekt der entsprechende Komplexe Wert für das ausgetauschte Submerkmal angegeben wird.
Bei manchen Merkmalen kommt es vor, dass ein oder mehrere der Submerkmale ebenfalls Subsubmerkmale besitzen. Eine solche Situation ist in der Abbildung 15 für das Submerkmal `etf:Asian.Asiatic` zu sehen. Diese Subsubmerkmale müssen natürlich ebenfalls angepasst werden. Jedoch ist es hier nicht notwendig, noch einmal das komplette Entwurfsmuster anzuwenden. Es reicht an dieser Stelle aus, für die Subsubmerkmale eine neue Subklasse von der bereits definierten Klasse anzulegen und jene zu Instanzen dieser Subklasse umzuwandeln. Die Verwendungen der Subsubmerkmale an den POI-Ressourcen müssen dann nach dem gleichen Prinzip, wie für die Submerkmale, angepasst werden. Das heißt, als Prädikat wird das neue Merkmal aus dem Entwurfsmuster verwendet und als Wert wird jeweils der entsprechende Komplexe Wert eingesetzt.

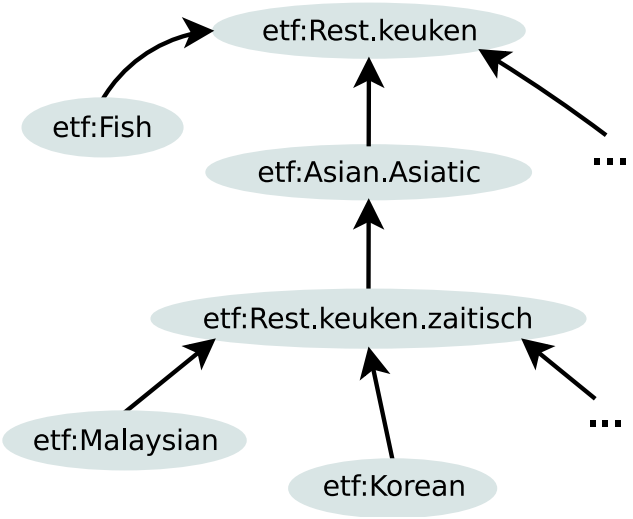


Abbildung 15: Ausschnitt aus der Hierarchie der Submerkmale von `etf:Rest.keuken`.

ANPASSUNGSSCHRITTE

1. Erzeugen einer neuen Klasse für die Komplexen Werte
2. Typisieren der Klasse als Subklasse von `skos:Concept`, falls es sich bei den Instanzen um abstrakte Komplexe Werte handelt
3. Ermittlung aller Submerkmale, für jedes Submerkmal wird ein neuer Komplexer Wert erzeugt, durch:
   a) Erzeugen einer neuen Ressource, welche Instanz der Klasse aus *1.* ist
   b) Kopieren der Bezeichner (`rdfs:label`)
   c) Kopieren der Kommentare (`rdfs:comment`)

   d) Kopieren der Beschreibungen (`dc:description`)
   e) Kopieren des Icons (`vakp:hasIcon`)
4. Erzeugen eines neuen Merkmals entsprechend dem Entwurfsmuster
5. Kopieren der Bezeichner (`rdfs:label`)
6. Kopieren der Kommentare (`rdfs:comment`)
7. Kopieren der Beschreibungen (`dc:description`)
8. Kopieren des Icons (`vakp:hasIcon`)
9. Ermittlung aller Tripel aus der Instanz-Ebene, in denen eines der alten Submerkmale vorkommt, für jedes dieser Tripel unter Beibehaltung des Subjekts (URI des POIs) wird ein neues Tripel erzeugt, durch:
   a) Austausch des alten Submerkmals gegen das neue Merkmal
   b) Einsetzen des entsprechenden Komplexen Wertes an die Objekt-Position
10. Löschen der alten Instanz-Tripel
11. Löschen der alten Submerkmale
12. Löschen des alten Merkmals

7.5 EVOLUTION ZU POLYVALENTEM MERKMAL

Im Kapitel über die Ontologie-Entwurfsmuster wurde bereits angesprochen, dass es in dem alten Schema Merkmale gibt, welche zur Angabe einer drei-stelligen Relation verwendet werden. Zur Erklärung soll an dieser Stelle das Merkmal `etf:Erkenningen` und dessen Submerkmale dienen. Wie diese Merkmale aktuell auf Instanz-Ebene genutzt werden, ist im Listing 19 zu sehen. Daraus ist zu entnehmen, dass auch hier die Submerkmale eher Komplexen Werten als Relationen entsprechen. Dagegen stellen die Werte der Literale aber einen Unterschied zu dem Beispiel aus dem vorherigen Abschnitt 7.4 dar. Es werden an dieser Stelle nicht ausschließlich boolesche Werte, repräsentiert durch eine „1“, angeknüpft, sondern auch Jahreszahlen. Durch diese Art der Verknüpfung soll demnach ausgedrückt werden, dass etwa die Anerkennung `etf:RegisteredMuseum` im Jahr 2001 erteilt wurde.

```
1 <http://vakantieland.nl/data/pois/cox> etf:ISO9002 "1" .
2 etf:ISO9002 rdfs:subPropertyOf etf:Erkenningen .
3 <http://vakantieland.nl/data/pois/paultetar>
    etf:Registeredmuseum "2000" .
```

```
4 <http://vakantieland.nl/data/pois/dongha> etf:Registeredmuseum
    "2001" .
5 etf:Registeredmuseum rdfs:subPropertyOf etf:Erkenningen .
6 ...
```

Listing 19: Verwendung der Submerkmale von `etf:Erkenningen` auf Instanz-Ebene.

Das alte Schema besitzt nur Merkmale, welche selbst einen Komplexen Wert darstellen und dazu verwendet werden, einen Einfachen Wert mit einem POI zu verknüpfen. Aus dieser Tatsache heraus kann die Evolution entsprechend vereinfacht werden. So muss zunächst ein neues Merkmal entsprechend dem Muster „Merkmale mit Komplexen Werten“ erzeugt werden. Die Submerkmale von `etf:Erkenningen` werden zu Komplexen Werten umgewandelt, in dem jeweils eine neue Instanz von der Klasse aus dem Entwurfsmuster gebildet wird. Bis zu diesem Schritt entspricht das dem gleichen Algorithmus wie dem aus Abschnitt 7.4. Ab dem Schritt, in dem die Instanzdaten angepasst werden, unterscheiden sich aber die beiden Vorgehen. So müssen zunächst neue Submerkmale von dem Merkmal aus dem Entwurfsmuster erzeugt werden. Diese Submerkmale bekommen eine URI bestehend aus der URI des Supermerkmals plus jeweils die Zeichenkette einer Ausprägung des zu modellierenden Einfachen Wertes. Für das Jahr 2001 würde beispielsweise das Merkmal `etf:acknowledge2001` erzeugt werden. Anschließend können die Instanzdaten entsprechend angepasst werden. Das heißt unter Beibehaltung des Subjekts (URI des POI) wird das alte Merkmal durch ein passendes neues Submerkmal ausgetauscht und an die Objekt-Position wird der entsprechende neue Komplexe Wert eingesetzt.

ANPASSUNGSSCHRITTE

1. Ausführen der Schritte aus „Evolution zu einem Merkmal mit Komplexen Werten“ mit Überspringen der Schritte 9, 10, 11 und 12
2. Ermittlung aller möglichen Ausprägungen des Einfachen Wertes, für jede Ausprägung:
   a) Erzeugen eines neuen Submerkmals für das Merkmal aus dem Entwurfsmuster mit der URI: URI des Merkmals aus dem Muster + Literalwert der Ausprägung
   b) Angabe eines Bezeichners mittels `rdfs:label` mit der Ausprägung als Wert
3. Ermittlung aller Tripel aus den Instanzdaten, in denen das alte Merkmal beziehungsweise eines der alten Submerkmale vorkommt, für jedes dieser Tripel unter Beibehaltung des Subjekts (URI des POIs) wird ein neues Tripel erzeugt, durch:
   a) Austausch des alten Merkmals durch ein entsprechendes neues Submerkmal des Hauptmerkmals

   b) Einsetzen des entsprechenden Komplexen Wertes an die Objekt-Position

4. Löschen der alten Instanz-Tripel
5. Löschen der alten Submerkmale
6. Löschen des alten Merkmals

7.6 LÖSCHEN EINES MERKMALS

In der Evolutionsstrategie wurde festgelegt, dass auf Instanz-Ebene nicht verwendete Merkmale aus dem Schema gelöscht werden sollen. Dies kann auf ein einzelnes Merkmal oder auf ein Merkmal und dessen Submerkmale zutreffen. Das Löschen ist in diesem Fall trivial. Zunächst müssen die eventuell vorhandenen und zu löschenden Submerkmale ermittelt werden. Für jedes dieser Submerkmale werden alle Tripel aus der Datenbank entfernt, wo das Submerkmal als Subjekt oder Objekt im Tripel auftaucht. Da die Merkmale nicht verwendet werden, müssen auch keine Tripel gelöscht werden, in denen eines der Merkmale an der Prädikat-Position steht. Zum Schluss wird das eigentliche Merkmal nach dem gleichen Prinzip wie für die Submerkmale gelöscht.

ANPASSUNGSSCHRITTE

1. Ermittlung aller Submerkmale, für jedes dieser Submerkmale:
   a) Löschen der Tripel, wo das Submerkmal an der Objekt-Position auftritt
   b) Löschen der Tripel, wo das Submerkmal an der Subjekt-Position auftritt
2. Löschen der Tripel, wo das Merkmal an der Objekt-Position auftritt
3. Löschen der Tripel, wo das Merkmal an der Subjekt-Position auftritt

7.7 VERSCHMELZEN VON MERKMALEN

Bei der Analyse des alten Schemas sind einige Merkmale aufgefallen, die in ihrer Semantik gleichbedeutend sind. So hat etwa das Merkmal `etf:1Flag` die gleiche Semantik wie `etf:1Star`. Diese jeweils gleichbedeutenden Merkmale sollen miteinander verschmolzen werden. Nach der Entscheidung, welches Merkmal beibehalten werden soll, müssen alle Instanzdaten entsprechend angepasst werden. Dazu müssen alle Verwendungen des zu verschmelzenden Merkmals durch das Merkmal, welches beibehalten werden soll, ersetzt werden. Anschließend kann das ersetzte

Merkmal gelöscht werden. Um die Verständlichkeit der Anpassungsschritte zu verbessern, wird das Merkmal, welches beibehalten werden soll, als Merkmal `A` bezeichnet. Das Merkmal, welches mit Merkmal `A` verschmolzen werden soll, wird Merkmal `B` genannt.

ANPASSUNGSSCHRITTE

1. Ersetzen des Merkmals `B` durch Merkmal `A` in allen Tripel, wo Merkmal `B` als Prädikat vorkommt
2. Löschen des Merkmals `B`

7.8 MARKIEREN EINES MERKMALS

Einige Merkmale aus dem alten Schema sind in ihrer Semantik fragwürdig. So stellt das Merkmal `etf:Fine`, welches ausdrücken soll, dass ein POI schön ist, im Sinne der Definition kein wirkliches Merkmal dar. Es kann eher als eine Art der Bewertung angesehen werden. Da diese Merkmale aber dennoch auf Instanz-Ebene verwendet und somit nicht bedenkenlos entfernt werden können, sollen sie für eine spätere Evolution markiert und temporär aus dem neuen Merkmals-Schema verschoben werden. Ein solches Merkmal kann wiederum selbst Submerkmale besitzen, die ebenfalls markiert und verschoben werden müssen. Zur Markierung der Merkmale wird ein spezielles Prädikat `etf:refactor` verwendet. Um nun ein Merkmal für eine spätere Evolution zu kennzeichnen, muss die Subprädikats-Relation zu dem Wurzelmerkmal `etf:feature` durch eine Relation zu `etf:refactor` ersetzt werden. Für etwaige Submerkmale muss dementsprechend das gleiche durchgeführt werden.

ANPASSUNGSSCHRITTE

1. Ermittlung aller Submerkmale, für jedes Submerkmal:
   a) Ersetzen des Wurzelmerkmals `etf:feature` durch `etf:refactor` bei der `rdfs:subPropertyOf`-Relation
2. Ersetzen des Wurzelmerkmals `etf:feature` durch `etf:refactor` bei der `rdfs:subPropertyOf`-Relation des Merkmals

8 IMPLEMENTIERUNG

Inhaltsangabe

In diesem Kapitel folgt nun auf Grundlage der Anforderungen und der definierten Ontologie-Entwurfsmuster aus den Kapiteln 5 und 6 die Implementierung der geforderten Funktionalitäten. Dazu werden zunächst die Anpassungen an dem semantischen Datenwiki OntoWiki und dessen Evolutionskomponente erläutert. Anschließend wird die Implementierung des Programms zur Durchführung der Evolution erklärt. Im letzten Teil werden die Anpassungen in der Vakantieland-Applikation an das neue Schema erläutert, sowie die Implementierung der fassetierten Suche für die touristische Merkmale vorgestellt.

8.1 ONTOWIKI–EVOLUTIONSKOMPONENTE

Damit die Evolutionskomponente von OntoWiki zur Ausführung der Evolution an der Ontologie genutzt werden kann, sind einige Änderungen notwendig gewesen. Welche Änderungen das sind und warum diese durchgeführt werden mussten, soll in diesem Abschnitt erläutert werden. Alle Änderungen wurden in Absprache mit dem Entwickler der Komponente direkt in den entsprechenden Entwicklungs-Zweig eingepflegt und sind somit Bestandteil der aktuellen Version (siehe Anhang A.3).

8.1.1 RPC-Schnittstelle

Die Evolutionskomponente wurde erst während der Bearbeitung dieser Arbeit fertiggestellt [Rieß, 2010] und besaß in dieser ersten veröffentlichten Version nur die Möglichkeit, Evolutionsmuster manuell über die Web-

Oberfläche auszuführen. Laut der Anforderung *FA010* soll es aber möglich sein, Evolutionsmuster durch den Aufruf einer RPC-Schnittstelle mit Hilfe eines entfernten, externen Programms vollautomatisch ausführen zu können. Eine solche RPC-Schnittstelle wurde durch die Anbindung der Evolutionskomponente an die bereits existierende `jsonrpc`-Komponente von OntoWiki realisiert. Es soll an dieser Stelle kurz auf die Funktionsweise dieser Komponente eingegangen werden, bevor die konkreten Implementierungsarbeiten vorgestellt werden.
Mit Hilfe der `jsonrpc`-Komponente ist es möglich, entfernte Aufrufe an OntoWiki auf Basis der JSON-RPC-Spezifikation[1] auszuführen. Ein solcher entfernter Methoden-Aufruf beinhaltet unter anderem ein in JSON serialisiertes Objekt mit drei Variablen:

`method`	Enthält den Namen der auszuführenden Funktion.
`params`	Die Menge von Variablen, welche an die Funktion als Argumente übergeben werden.
`id`	Ein beliebiger Identifikator, welcher zur späteren Zuordnung der Antwort zu dieser Anfrage dient.

Ist der Methodenaufruf beendet, so wird ebenfalls ein in JSON serialisiertes Objekt mit den folgenden Variablen zurückgeliefert:

`result`	Enthält das Objekt, welches durch die Funktion zurückgegeben wurde.
`error`	Enthält ein Fehlerobjekt, falls bei der Ausführung ein Fehler aufgetreten ist.
`id`	Der selbe Identifikator, welcher durch den Aufruf übergeben wurde.

Das Listing 20 zeigt, wie eine solche Anfrage an die Komponente gestellt werden kann und wie die entsprechende Antwort aussieht.

```
1  $ wget -q -O - --post-data='{"method": "listServer", "params":
       [], "id": 33}' http://localhost/ontowiki/jsonrpc/meta
2
3  {
4   "result":[
5    {"name":"meta","description":"methods to query the json
        service itself"},
6    {"name":"store","description":"methods to manipulate and query
        the store"},
7    {"name":"model","description":"methods to manipulate and query
        a specific model"}
8   ],
9   "id":"33"
10 }
```

Listing 20: Beispiel-Anfrage an die `jsonrpc`-Komponente von OntoWiki, zur Auflistung aller verfügbaren Server.

1 http://json-rpc.org/

Um nun einen Server für die Evolutionskomponente hinzuzufügen, wurde eine neue `Wrapper`-Klasse, `evolutionJsonrpcWrapper`, im Ordner der `jsonrpc`-Komponente erzeugt. Diese Klasse dient als Server für die Evolutionskomponente, welche im Konstruktor dieser Klasse initialisiert wird. Momentan steht `execPattern()`, zur Ausführung von Evolutionsmustern, als einzige Methode dieses Servers zur Verfügung. Wird nun diese Funktion durch eine entfernte Anfrage aufgerufen, so wird zunächst geprüft, ob alle benötigten Parameter durch die Anfrage mitgeliefert werden. Diese benötigten Parameter müssen durch die `params`-Variable des mitgelieferten JSON-Objekts gebunden sein. Folgende drei Parameter werden für die Ausführung eines Evolutionsmusters benötigt:

`pattern`	Das Evolutionsmuster, welches ausgeführt werden soll, im URL-kodierten JSON-Format.
`graph`	Der Standard-Graph, welcher verwendet wird, wenn keine Graph-Angaben im Evolutionsmuster gegeben sind.
`variables`	Eine in JSON-kodierte Liste aus Paaren von Variablen mit Werten. Die Variablennamen müssen dabei denen aus der Menge `V` des Evolutionsmusters entsprechen.

Treten Fehler bei der Ausführung der Methode auf, so wird eine entsprechende Fehlermeldung geworfen. Damit der neue Server verwendet werden kann, musste dieser abschließend noch in die Liste der verfügbaren Server der Klasse `metaJsonrpcWrapper` hinzugefügt werden. Durch diese Erweiterung ist es nun möglich, über eine RPC-Schnittstelle die Evolutionskomponente anzusprechen. Eine solche beispielhafte Anfrage ist in Listing 21 zu sehen.

```
1  {
2   "method": "execPattern",
3   "params":
4    [{
5     "pattern":"<muster>",
6     "graph":"http://vakantieland.nl/catalogue",
7     "variables":
          {"class":"http://vakantieland.nl/catalogue/features/Sport"}
8    }],
9   "id": 1
10 }
```

Listing 21: Beispiel für ein JSON-Objekt, welches zur Ausführung eines Evolutionsmusters an den Server der Evolutionskomponente übertragen werden muss. <`muster`> steht dabei für das URL-kodierte Evolutionsmuster, ebenfalls im JSON-Format.

8.1.2 Optimierungen

Während der Ausführung der ersten erstellten Evolutionsmuster kam es zu Fehlern durch die Evolutionskomponente. Diese Fehler äußerten sich in zwei unterschiedlichen Formen.

Einschränkung von SELECT

So kam es einerseits zu einem Fehler durch einen zu großen Rückgabewert, verursacht durch bestimmte SPARQL-Anfragen an die Datenbank. Nach der Analyse der `WHERE`-Klauseln der entsprechenden Evolutionsmuster konnte an dieser Stelle keine Optimierung mehr durchgeführt werden. Aus diesem Grund wurde die Evolutionskomponente genauer untersucht, um den Fehler zu beseitigen. Der erste Schritt bestand darin, die SPARQL-Anfrage, welche in der Klasse `BasicPattern` aus den Informationen eines `Basic Evolution Patterns` zusammengesetzt wird, zu analysieren. Es stellte sich heraus, dass diese Zusammensetzung nicht optimal umgesetzt war. So wurde für die `SELECT`-Anfrage generell das Akronym „*“ verwendet, wodurch alle möglichen Werte sämtlicher Variablen einer Anfrage in der Ergebnismenge gebunden werden. Enthält eine Anfrage viele verschiedene Variablen, so kann die Verwendung von „*“ schnell zu einer sehr großen Ergebnismenge führen. Ein kurzes Beispiel soll diesen Zusammenhang genauer erläutern. Es sei eine `SELECT`-Anfrage mit zwei verwendeten Variablen `A` und `B` gegeben. Für `A` gibt es fünf mögliche Belegungen und für `B` zwei. Wird nun das Akronym „*“ zur Selektion der Variablen verwendet, so enthält die Ergebnismenge insgesamt 10 Tupel, entstanden aus der Kombination aller möglichen Belegungen von `A` und `B`. Sind aber nur die Belegungen von `A` relevant, kann zwar auch die Ergebnismenge aus der ersten Variante verwendet werden, jedoch lässt sich durch die Einschränkung der Selektion auf `A` die Menge der Tupel und somit auch die Größe der Ergebnismenge verkleinern. Wird nur `A` für die Ergebnismenge selektiert, so enthält diese lediglich fünf Tupel, wobei jedes einzelne Tupel auch nur einen Wert enthält, was fünf Zellen in der Ergebnismenge darstellt. Bei der ersten Variante bestand jedes Tupel aus einem Paar von Werten, was letztendlich zu 20 Zellen führt. Durch die richtige Selektion der Variablen einer Anfrage kann also die Größe der Ergebnismenge entscheidend beeinflusst werden. Aus dieser Überlegung heraus entstand der Bedarf, die Zusammensetzung der `SELECT`-Anfrage zu optimieren. Dazu musste ermittelt werden, auf welche Menge von Variablen die Ergebnismenge reduziert werden kann, damit die Komponente weiterhin korrekt funktioniert. Insgesamt wird in der Evolutionskomponente zwischen zwei Arten von Variablen unterschieden. Zum einen die SPARQL-Variablen, welche durch ein führendes „?“ gekennzeichnet sind und zum anderen die Variablen aus der Menge `V` eines Evolutionsmusters, welche durch ein „%“-Zeichen vor und nach dem Variablennamen innerhalb der Klauseln markiert werden. Da die Evolutionskomponente nur auf den Variablen der Menge `V` arbeitet, können somit die SPARQL-Variablen aus der Selektion für die Ergebnismenge ausgeschlossen werden. Die Variablen aus `V` können nun aber noch weiter enstprechend ihrem Typ unterteilt werden. An dieser Stelle reicht eine Unterteilung in temporäre (vom Typ `TEMP`) und andere Variablen (zum Beispiel `RESOURCE`, `LITERAL` und so weiter) aus. Sämtliche Variablen, die nicht vom Typ `TEMP` sind, werden bereits durch die Komponente mit festen Werten belegt. Nur die temporären Variablen erhalten ihre möglichen Belegungen erst durch die `SELECT`-Anfrage. So-

mit sind auch nur diese für eine Selektion durch die Ergebnismenge relevant. Durch eine Einschränkung der `SELECT`-Anfrage in der Methode `executeSelect()` der Klasse `BasicPattern` auf diesen Typ von Variablen konnte letztendlich das Problem einer zu großen Ergebnismenge, welche zu einem Fehler führte, beseitigt werden.

Verwendung von DISTINCT

Der zweite Fehler äußerte sich durch zu viele Tripel innerhalb der generierten SPARQL/Update-Anfragen aus der Menge `U` eines `Basic Evolution Patterns`. Eine genauere Analyse ergab, dass bestimmte Tripel mehrfach in den `INSERT`- beziehungsweise `DELETE`-Anfragen auftraten, wodurch der besagte Fehler versursacht wurde. Dieses Problem konnte auf die Tatsache zurückgeführt werden, dass bestimmte Tupel aus der Ergebnismenge der `SELECT`-Anfrage ebenfalls mehrfach auftraten und somit beim Tupel-weise Abarbeiten die selben Tripel für die `INSERT-/DELETE`-Anfragen wiederholt generiert wurden. Um das mehrfache Auftreten des selben Tupels in der Ergebnismenge einer `SELECT`-Anfrage zu vermeiden, bietet SPARQL den Modifikator `DISTINCT` an. Durch ein Hinzufügen von `DISTINCT` zur `SELECT`-Anfrage in der Methode `executeSelect()`, wiederum in der Klasse `BasicPattern`, konnte auch dieser Fehler beseitigt werden.

8.2 EVOLUTIONSPROGRAMM

Zur Durchführung der Evolution an der eTourismus-Ontologie wurde ein spezielles Programm entwickelt, welches ausschließlich für den Vakantieland-Anwendungsfall konzipiert ist. Es dient in der aktuellen Version (siehe Anhang A.4) zur Evolution des Schemas der touristischen Merkmale. Diese Evolution der Merkmale findet direkt nach der semantischen Annotation der Daten statt. Die Ausführung fügt sich demnach direkt an das Ende des Installationsprozess einer Vakantieland-Instanz an.

Das Evolutionsprogramm wurde entsprechend den Anforderungen komplett in PHP implementiert. Es lässt sich, bis auf eine spezielle OntoWiki-Instanz, ohne Installation zusätzlicher Programme auf jedem System ausführen, auf dem die Vakantieland-Applikation lauffähig ist. Voraussetzung für die OntoWiki-Instanz ist, dass diese Zugriff auf die genutzte Ontologie hat und über die in Abschnitt 8.1 beschriebene Komponente und Funktionalitäten verfügt. Ausgeführt wird das Evolutionsprogramm mit Hilfe der PHP-Laufzeitumgebung. Der entsprechende Aufruf unter einem UNIX-basierten System ist in Listing 22 zu sehen. Wird das Programm gestartet, so wird zunächst die Konfiguration (siehe Abschnitt 8.2.2) eingelesen und ein neues Objekt der Klasse `EvolutionWrapper` erzeugt. Die Erläuterung zu dieser und weiteren Klassen, sowie dem generellen Aufbau des Programms, folgt im Abschnitt 8.2.1. Anhand der Konfiguration und mit Hilfe des `EvolutionWrappers` wird dann die Evolution durchge-

führt. Zu Beginn werden der Ontologie ein paar neue schematische Ressourcen hinzugefügt. Zum einen das `etf:refactor`-Prädikat, welches zur Markierung von Merkmalen dient, die durch eine spätere Evolution bearbeitet werden sollen. Zum anderen werden die drei neuen Merkmals-Klassen `etc:SimpleValueFeature`, `etc:ComplexValueFeature` und `etc:PolyvalentFeature` angelegt, welche in den entsprechenden Ontologie-Entwurfsmustern definiert wurden. Anschließend findet die eigentliche Anpassung der Merkmale statt, indem die Konfiguration für die Evolution der Merkmale sektionsweise abgearbeitet wird. Tritt hierbei ein Fehler auf, so bricht das Programm sofort ab und die Beschreibung des Fehlers wird ausgegeben. Läuft das Programm ohne einen Fehler durch, ist die Evolution erfolgreich durchgeführt.

```
1 $ php {root}/admin/evolution/e.php
```

Listing 22: Befehl zur Ausführung des Evolutionsprogramms. `root` muss durch das Installationsverzeichnis der Vakantieland-Instanz ersetzt werden.

8.2.1 Aufbau

Die Struktur des Evolutionsprogramms ist sehr einfach gehalten. Es besteht zum einen aus zwei Klassen, einem PHP-Skript zur Ausführung und zwei Konfigurationsdateien (siehe Abschnitt 8.2.2). Weiterhin sind sämtliche definierten Evolutionsmuster (siehe Abschnitt 8.2.3) Bestandteil des Programms. Die Abbildung 16 zeigt den Aufbau des Programms in Form eines Diagramms.
Die Klasse `UrlHandler` dient zur Ausführung der HTTP POST-Anfragen an die `jsonrpc`-Komponente von OntoWiki und implementiert damit die Anforderung *FA021*. Die Methode `postRequest()` stellt die einzige Funktion dieser Klasse dar. Über die drei Parameter `$url` (anzufragende Adresse), `$data` (zu übertragende Daten) und `$optional_headers` (optionale Header-Angaben) werden die benötigten und zu übermittelnden Informationen übergeben. Nach Beendigung der Anfrage wird der Rückgabewert (HTTP Response[2]) an die aufrufende Funktion zurückgegeben.
Die `EvolutionWrapper`-Klasse implementiert die in Kapitel 7 vorgestellten Algorithmen zur Anpassung des Merkmals-Schemas. Dabei ist jeder Algorithmus durch eine spezielle Methode umgesetzt. In jeder dieser Methoden werden in der Reihenfolge der definierten Anpassungsschritte entsprechende Evolutionsmuster ausgeführt, welche wiederum jeweils einen speziellen Anpassungsschritt umsetzen. In der Tabelle 6 sind die möglichen Optionen für die einzelnen Methoden aufgelistet, mit deren Hilfe die Ausführung bestimmter Anpassungsschritte beeinflusst werden kann. Die Optionen werden durch eine Komma-separierte Liste angegeben, falls mehrere verwendet werden sollen. Die Bedeutungen der

2 http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6

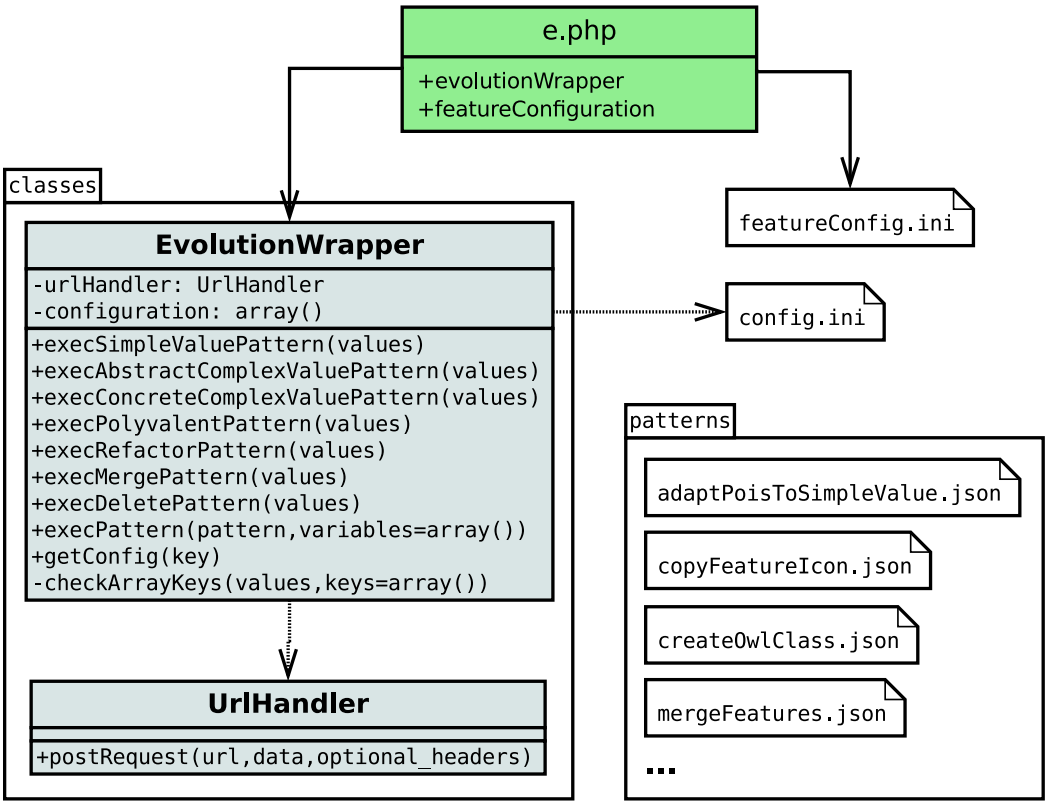


Abbildung 16: Aufbau des Evolutionsprogramms.

einzelnen Optionen können der Tabelle 8 im Abschnitt 8.2.2 entnommen werden.

8.2.2 Konfiguration

Die Konfiguration des Evolutionsprogramms erfolgt über die beiden Initialisierungsdateien `config.ini` und `featureConfig.ini`.
In der `config.ini` sind allgemeine Einstellungen festgelegt. Die Angaben zur Adresse der RPC-Schnittstelle der Evolutionskomponente, `owUrl`, sowie die dafür benötigten Account-Daten, `owUser` und `owPass`, müssen vor dem Start des Evolutionsprogramms an die jeweiligen Gegebenheiten angepasst werden. Zur Ausführung der Evolutionsmuster benötigt der angeführte Account Lese- und Schreibrechte für den Schema- und Instanzgraphen der eTourismus-Ontologie. Alle weiteren Einstellungen bedürfen im Allgemeinen keiner Anpassung und werden deshalb an dieser Stelle nicht weiter erläutert.
Die Konfiguration für die Evolution der Merkmale befindet sich in der Datei `featureConfig.ini`. Dabei spielt die Reihenfolge der dort enthaltenen Angaben eine entscheidende Rolle, da die unterschiedlichen Anpassungen der einzelnen Merkmale sich gegenseitig beeinflussen können. Die Begründung für diese Reihenfolge und die Bedeutungen

Methode	Optionen
execSimpleValuePattern()	noLabels, noDelete, noSubjectDelete
execAbstractComplex-ValuePattern()	noLabels, noDelete, noFeature
execConcreteComplex-ValuePattern()	noLabels, noDelete, noFeature
execPolyvalentPattern()	noLabels, noDelete, noFeature, concreteComplexValue, abstractComplexValue
execRefactorPattern()	subFeatures
execMergePattern()	—
execDeletePattern()	subFeatures

Tabelle 6: Methoden der Klasse `EvolutionWrapper`, welche die Algorithmen zur Evolution der Merkmale implementieren und deren mögliche Optionen.

der einzelnen Schlüssel-Wert-Paare soll nun anhand der Konfiguration, welche im Listing 23 angegeben ist, für das Merkmal `etf:Rest.keuken` und dessen Submerkmale erläutert werden.

```
1  [Rest.keuken]
2  patternType = "abstractComplexValue"
3  feature = "vakpf:Rest.keuken"
4  newFeature = "vakpf:hasCuisine"
5  newClass = "etc:Cuisine"
6  uriSuffix = ""
7  options = ""
8
9  [Rest.keuken.europees]
10 patternType = "abstractComplexValue"
11 feature = "vakpf:Rest.keuken.europees"
12 newFeature = "vakpf:hasCuisine"
13 newClass = "etc:EuropeanCuisine"
14 superClass= "etc:Cuisine"
15 uriSuffix = ""
16 options = "noFeature"
17
18 [Rest.keuken.ziatisch]
19 patternType = "abstractComplexValue"
20 feature = "vakpf:Rest.keuken.ziatisch"
21 newFeature = "vakpf:hasCuisine"
22 newClass = "etc:AsianCuisine"
23 superClass= "etc:Cuisine"
24 uriSuffix = ""
25 options = "noFeature"
```

Listing 23: Konfiguration für das Merkmal `etf:Rest.keuken` und dessen Submerkmale.

Wie in Listing 23 zu sehen, ist jedes Merkmal, welches einer Evolution unterzogen werden soll, durch eine eigene Sektion innerhalb der Kon-

figurationsdatei angegeben. Der Schlüssel `patternType` legt dabei den anzuwendenden Algorithmus zur Evolution dieses Merkmals fest. Die möglichen Werte für diesen Schlüssel können der Tabelle 7 entnommen werden. Das Merkmal `Rest.keuken` und die beiden Submerkmale `Rest.keuken.europees` und `etf:Rest.keuken.ziatisch` werden laut der Angabe zu Merkmalen mit abstrakten Komplexen Werten umgewandelt.

Wert	Algorithmus
simpleValue	Evolution zu einem Merkmal mit Einfachen Werten.
concreteComplexValue	Evolution zu einem Merkmal mit konkreten Komplexen Werten.
abstractComplexValue	Evolution zu einem Merkmal mit abstrakten Komplexen Werten.
polyvalent	Evolution zu einem Polyvalenten Merkmal.
merge	Verschmelzen von zwei Merkmalen.
refactor	Markieren eines Merkmals.
delete	Löschen eines Merkmals.

Tabelle 7: Mögliche Werte für den Schlüssel `patternType`.

Mit dem Schlüssel `feature` wird die genaue URI des zu evolutionierenden Merkmals genannt, `newFeature` gibt die URI des neuen Merkmals an. Da für die Komplexen Werte eine neue Klasse erzeugt werden muss, wird die URI für diese mit Hilfe des Schlüssels `newClass` angegeben. Handelt es sich um abstrakte Komplexe Werte, wird diese Klasse vom System automatisch als Subklasse von `skos:Concept` deklariert. Soll zusätzlich, zur besseren Hierarchisierung der Komplexen Werte untereinander, eine Superklasse festgelegt werden, wird die URI für diese Klasse über den Schlüssel `superClass` angegeben. So stellen für das Beispiel aus dem Listing 23 die Klassen `etc:EuropeanCuisine` und `etc:AsianCuisine` jeweils eine Spezialisierung der Klasse `etc:Cuisine` dar. Über den Schlüssel `uriSuffix` kann eine Zeichenkette angegeben werden, die bei der Erstellung der URIs für die konkreten oder abstrakten Komplexen Werte angehängt wird, um so eventuell zwei unterschiedliche Ressourcen mit identischen URIs zu vermeiden. Mit Hilfe des Schlüssels `options` lassen sich die Schritte der Algorithmen zur Evolution beeinflussen. Für das hier vorgestellte Beispiel ist für die beiden Merkmale `Rest.keuken.europees` und `etf:Rest.keuken.ziatisch` die Option „noFeature“ angegeben. Dadurch wird die Erzeugung des neuen Merkmals übersprungen, da `etf:hasCuisine` bereits bei der Evolution von `etf:Rest.keuken` erzeugt wurde. Dennoch wird `etf:hasCuisine` dazu verwendet, um die Komplexen Werte mit den POIs zu verknüpfen. In der Tabelle 8 sind die weiteren möglichen Werte für den Schlüssel `options` mit einer kurzen Erläuterung

angegeben.

Wert	Bedeutung
noSubjectDelete	Die Tripel, in denen das alte Merkmal als Subjekt auftritt, werden nicht gelöscht.
noFeature	Es wird kein neues Merkmal erzeugt.
noLabels	Die Bezeichner des alten Merkmals werden nicht zum neuen Merkmal kopiert.
subFeatures	Die Submerkmale werden bei der Evolution des Merkmals ebenfalls bearbeitet.
noDelete	Es werden keine Tripel gelöscht.

Tabelle 8: Mögliche Werte für den Schlüssel `options`.

8.2.3 Evolutionsmuster

Sämtliche Evolutionsmuster basieren auf der Spezifikation der Evolutionskomponente von OntoWiki und sind im JSON-Format notiert. Die Erzeugung vieler einzelner, aber dafür weniger komplexer Evolutionsmuster hatte mehrere Gründe. So kam es trotz der Optimierungen an der Evolutionskomponente immer noch zu den genannten Problemen auf Grund zu großer Ergebnismengen. Durch die Aufteilung in mehrere kleinere Evolutionsmuster konnte die jeweils zu bearbeitende Datenmenge deutlich verkleinert werden, wodurch diese Fehler vermieden werden konnten. Jedoch muss an dieser Stelle darauf hingewiesen werden, dass bei einer eventuell größer werdenden Datenmenge, welche einer Evolution unterzogen werden soll, die besagten Probleme wieder auftreten können. Ein weitere Vorteil dieser Auftrennung der Evolutionsmuster besteht in der Flexibilität der Anwendung und in der Wiederverwendbarkeit. Um zum Beispiel die verschiedenen Konfigurationsmöglichkeiten zu unterstützen, ist es notwendig, dass bestimmte Anpassungsschritte der einzelnen Algorithmen aus Kapitel 7 ausgelassen werden können. Wäre jeder Algorithmus in jeweils einem `Compound Evolution Pattern` bestehend aus vielen `Basic Evolution Pattern` realisiert worden, hätten einzelne Schritte nicht ohne Weiteres ausgelassen werden können. Die einzige Möglichkeit hätte darin bestanden, jedes `Compound Evolution Pattern` zu kopieren und das für den Anpassungsschritt entsprechende `Basic Evolution Pattern` zu entfernen, was aber letztendlich in einer ähnlich großen Menge an Evolutionsmustern resultiert hätte, die zudem deutlich schlechter wartbar gewesen wäre. Ein letzter Grund für die Einteilung in viele kleinere Evolutionsmuster ist die bessere Unterstützung bestimmter Phasen des Evolutionsprozesses (siehe Abschnitt 4.1.1). So wird vor allem das Verständnis der dritten Phase, die Semantik der Änderungen, durch klei-

nere Evolutionsmuster mit demzufolge weniger Änderungsoperationen erleichtert.
Es folgt nun beispielhaft die kurze Vorstellung eines der erstellten Evolutionsmuster. An diesem soll erläutert werden, wie die Evolution der Merkmale im Detail vonstatten geht. Das Beispiel ist in Listing 24 zu sehen und dient zur Umsetzung der Anpassungsschritte 6a und 6b aus dem Algorithmus zur „Evolution zu einem Merkmal mit Einfachen Werten“ (siehe Abschnitt 7.3). Wie dem Listing 24 zu entnehmen ist, besteht die Menge `V` aus sechs Variablen. Durch `feature` wird die URI des neuen Merkmals gebunden, `oldFeat` enthält die URI des alten Merkmals. Der Datentyp wird durch die Variable `datatype` gebunden. Da es sich bei diesem Beispiel um eine Anpassung der Instanzdaten handelt, muss die URI des entsprechenden Instanzgraphen an die Variable `graph` übergeben werden. Die Werte der beiden temporären Variablen `poi` und `value` werden erst durch die Ergebnismenge einer SPARQL `SELECT`-Anfrage, bestimmt durch die `WHERE`-Klausel der Menge `S`, belegt. In diesem Falle werden durch `S` alle POIs und die jeweils durch das alte Merkmal verknüpften Datenwerte bestimmt. Für jedes Tupel dieser Ergebnismenge wird dann, angegeben durch die Menge `U`, ein neues Tripel erzeugt, welches als Subjekt die POI-URI, als Prädikat das neue Merkmal und als Objekt den korrekt getypten Datenwert enthält. Diese erzeugten Tripel werden dann anschließend mit Hilfe einer `INSERT`-Anfrage dem Instanzgraphen hinzugefügt.

```
{
 "label":"Transform Simple Value feature",
 "desc":"Transforms a Simple Value feature.",
 "subPattern":[{
  "label":"Transform POI triples",
  "desc":"Transforms all triples where the old feature is used
     to the new schema.",
  "V":[
   {"name":"feature","bound":false,"type":"RESOURCE","desc":""},
   {"name":"oldFeat","bound":false,"type":"RESOURCE","desc":""},
   {"name":"datatype","bound":false,"type":"RESOURCE","desc":""},
   {"name":"graph","bound":false,"type":"GRAPH","desc":""},
   {"name":"poi","bound":false,"type":"TEMP","desc":""},
   {"name":"value","bound":false,"type":"TEMP","desc":""}
  ],
  "S":"{ %poi% %oldFeat% %value% }",
  "U":[{
   "pattern":"%poi% %feature%
      createTypedLiteral(%value%,%datatype%) %graph%",
   "type":"insert"}
  ]
 }]
}
```

Listing 24: Das Evolutionsmuster `adaptPoisToSimpleValue.json` im JSON-Format.

Alle weiteren erzeugten Evolutionsmuster mit einer kurzen Beschreibung können der Tabelle 11 im Anhang A.2 auf der Seite 95 entnommen werden.

8.3 ANWENDUNGSFALL VAKANTIELAND

Die Vakantieland-Applikation baut bekanntlich direkt auf der in dieser Arbeit einer Evolution unterzogenen Ontologie auf. Die Ontologie wird dabei zum Lesen und Speichern der in der Applikation angezeigten touristischen Informationen genutzt. Durch eine Veränderung des Schemas der Merkmale mussten demzufolge auch Anpassungen an Vakantieland durchgeführt werden, was zur fünften Phase des Evolutionsprozesses, der Änderungspropagierung, gehört. Im Anschluss daran wird die neu integrierte Filterfunktion für die touristischen Merkmale vorgestellt.

8.3.1 Anpassungen an das neue Merkmals-Schema

Nach der Ausführung der Anpassungen am Schema der Merkmale und einer damit einhergehenden Veränderung der Struktur der Daten in der Ontologie, wurden durch die Vakantieland-Applikation einige Informationen nicht mehr korrekt ausgewertet und demzufolge auch falsch angezeigt. Im konkreten Fall betraf das die Detailsseite zur Anzeige aller Informationen für einen POI. Die Daten im Reiter für die Darstellung der Merkmale wurden nicht mehr korrekt angezeigt, was in der Abbildung 17 zu sehen ist.
Um die korrekte Darstellung der Merkmale auf der Detailseite wieder herzustellen, musste die Methode `createPropertyTableLines()` der Klasse `View_Component_PropertyTable` an das neue Schema der Merkmale angepasst werden. Durch die Evolution beziehungsweise die neu definierten Ontologie-Entwurfsmuster wurden auch drei neue Merkmalsklassen eingeführt. Anhand dieser Klassen kann nun auf Seiten der Applikation ausgewertet werden, wie ein Merkmal einer dieser Klassen dargestellt werden soll. Da aber bis zu dieser Arbeit keine Einteilung in Klassen für Merkmale existierten, gab es auch keine Möglichkeit den Typ eines Merkmals zu bestimmten. Aus diesem Grund musste die Klasse `Model_Wrapper_Features` erweitert werden. Durch eine Anpassung der SPARQL-Anfrage in der Methode `retrieveFeatures()` wird nun auch der Typ eines Merkmals abgefragt. Mit Hilfe der neuen Methode `getType()` kann dann der Typ eines Merkmals bestimmt werden. Die Abbildung 18 zeigt die neue, an das Schema angeglichene, Detailseite. Die Merkmale und die zugehörigen touristischen Werte werden wie zuvor weiterhin in einer Tabellen-artigen Anordnung dargestellt, jedoch in alphabetischer Reihenfolge, um dem Nutzer das Auffinden eines Merkmals zu erleichtern.

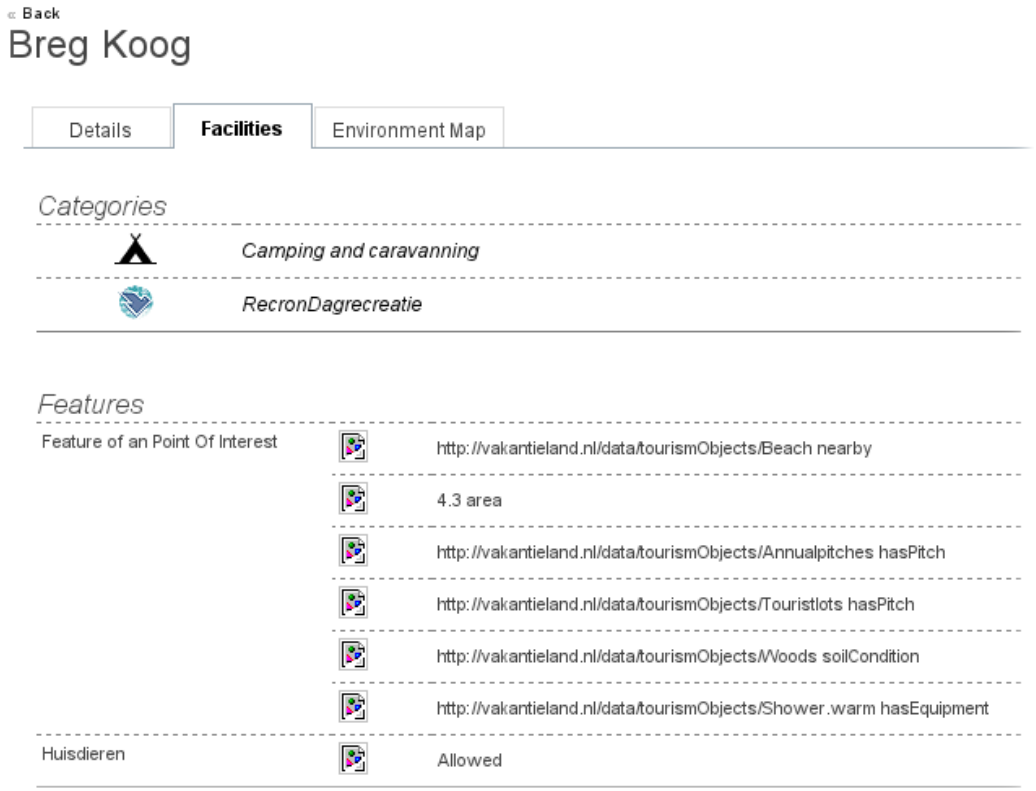


Abbildung 17: Fehlerhafte Anzeige der Merkmale nach der Evolution ohne Anpassung der Vakantieland-Applikation.

8.3.2 Fassetierte Suche für Merkmale

Eine Filtermöglichkeit nach den touristischen Merkmalen war bereits vor dieser Arbeit schon einmal angedacht, weswegen auch bereits einige Vorkehrungen innerhalb der Quellcode-Basis von Vakantieland vorhanden waren. Doch wegen der mangelhaften Modellierung der Merkmale wurde dies nie umgesetzt. Durch die Evolution der Merkmale zu einem neuen, klar definierten Schema wurde nun als konkretes Anwendungsbeispiel eine solche fassetierte Suche in die Startseite der Vakantieland-Applikation integriert, wie in Abbildung 19 zu sehen.
Wie schon erwähnt, waren die Methoden zur Integration der Suche bereits vorbereitet. So mussten lediglich zwei neue Klassen erzeugt und die bereits existierenden Methoden entweder angepasst oder implementiert werden. Eine Übersicht zu diesen Methoden bietet die Tabelle 9. Durch die Klasse `View_Component_Panel_Properties` wird die komplette GUI-Komponente (rot markierter Teil aus Abbildung 19) bereitgestellt, wobei die einzelnen Listenelemente mit den Auswahlboxen mit Hilfe der Klasse `View_Component_Select_Properties` erzeugt werden. Zur Anzeige der neuen Komponente auf der Startseite war eine Integration in das dafür zuständige GUI-Template `content.php` notwendig. Da es sich bei den verschiedenen Filterboxen auf der Startseite jeweils um eigenständi-

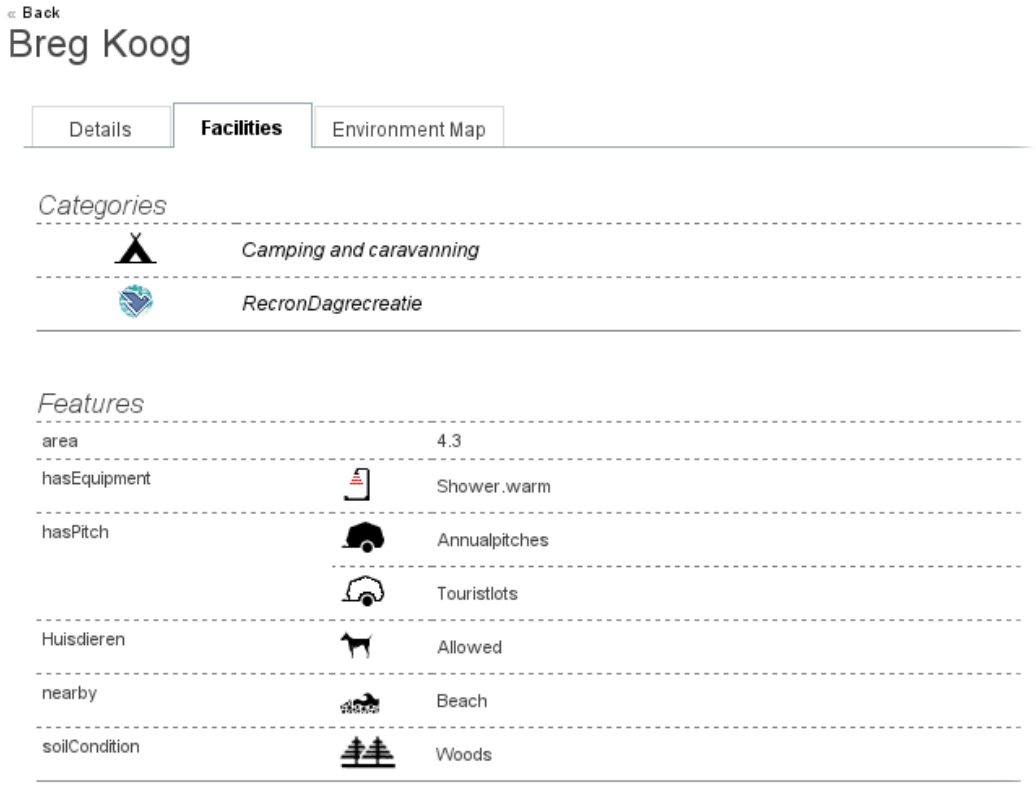


Abbildung 18: Die an das neue Schema angepasste Anzeige der touristischen Merkmale für einen POI.

ge HTML-Formulare handelt, werden beim Absenden der Auswahl einer Filterbox die getätigten Filtereinstellungen der anderen Filterboxen standardmäßig ignoriert. Dies lässt ohne den Einsatz von Javascript auch nicht ändern. Um dennoch die gemeinsame Nutzung der einzelnen Filterboxen zu gewährleisten, um unter anderem die Anforderung *FA041* zu erfüllen, werden spezielle versteckte HTML-Input-Elemente verwendet. Diese Formular-Elemente werden durch den Browser nicht dargestellt, können aber zum Binden von Werten beim Absenden eines Formulars genutzt werden. Dadurch wird zwar nicht das gleichzeitige Ändern von Filterkriterien in mehreren Filterboxen ermöglicht, jedoch bleiben die einmal getätigten Filtereinstellungen gespeichert, wodurch beim Absenden der gewählten Kriterien einer Filterbox, die zuvor getätigten Kriterien einer anderen Filterbox nicht verloren gehen. Damit nun die Filterkriterien aus der Filterbox für die Merkmale über mehrere Anfragen hinweg gespeichert werden, musste die Methode `renderHiddenFields()` der Klasse `Core_View` um die Unterstützung für die ausgewählten Merkmalskriterien erweitert werden.
Es soll nun noch kurz auf die Funktionsweise der neuen fassetierten Suche für die touristischen Merkmale eingegangen werden. Mit Hilfe der Auswahlmenüs können für die jeweiligen Merkmale passende Werte ausgewählt werden. Hinter den einzelnen Werten steht zusätzlich die Anzahl,

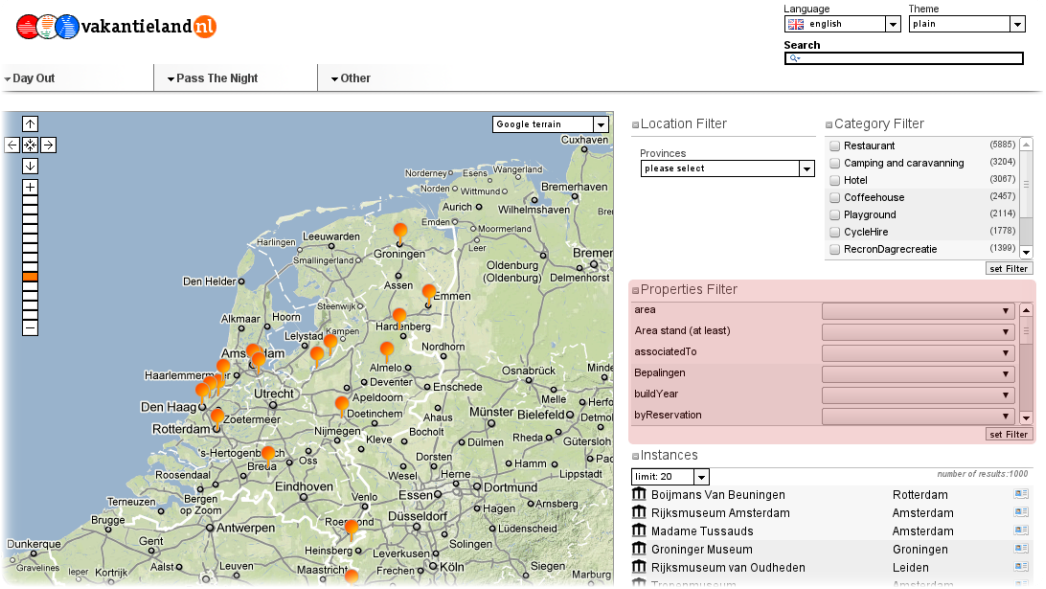


Abbildung 19: Die neue Startseite von Vakantieland mit der integrierten Suche für die touristischen Merkmale (rote Markierung).

wie viele POIs über diese Merkmals-Werte-Kombination verfügen, sortiert nach der Häufigkeit. Die Merkmale innerhalb der Liste der Filterbox sind alphabetisch sortiert, um dem Nutzer die Suche nach einem Merkmal so einfach wie möglich zu gestalten. Wie in der Abbildung 20 zu sehen, werden nach dem Absenden die ausgewählten Merkmale zu Beginn der Liste angezeigt. Über die Kontrollbox kann die getätigte Auswahl wieder rückgängig gemacht werden.

Methode (Klasse)	Erläuterung
getFacettedFilter() (Model_Wrapper_Classes)	Bestimmung der POI-Klassen oder Merkmale, welche entsprechend den bisher getätigten Filtereinstellungen noch zur Auswahl stehen.
getPropertiesFilter() (Core_Controller)	Aufbereitung der getätigten Merkmals-Filterkriterien aus dem HTML-Request-Objekt.
createPropertiesTriplePattern() (Model_Wrapper_PointOfInterests)	Erstellung der Tripel-Muster für die Merkmale, welche für die SPARQL-Anfrage zur Ermittlung der auf die Filterkriterien zutreffenden POIs genutzt werden.

Tabelle 9: Methoden, die zur Integration der fassetierten Suche für die Merkmale erweitert beziehungsweise implementiert werden mussten.

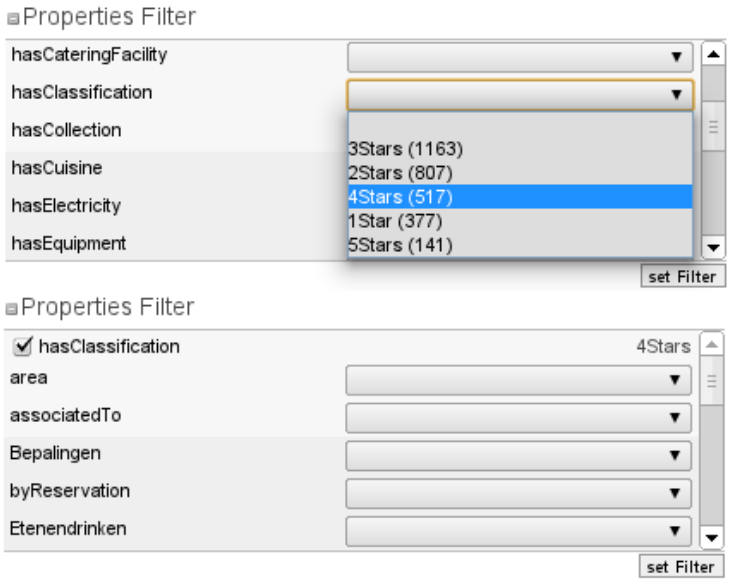


Abbildung 20: Die obere Bildhälfte zeigt die Merkmals-Filterbox bei der Auswahl eines Wertes für ein Merkmal. In der unteren Bildhälfte ist die Filterbox nach dem Absenden der Auswahl zu sehen.

9 ZUSAMMENFASSUNG

Inhaltsangabe

Das Semantic Web stellt eine Erweiterung des bisherigen Dokumenten-orientierten World Wide Web dar. Durch die semantische Annotation soll eine bessere Verarbeitung der bereitgestellten Informationen ermöglicht werden. Die dafür entwickelten Standards und Technologien, welche auch für diese Arbeit von Bedeutung sind, wurden im Kapitel 2 vorgestellt.

Für den eTourismus bietet das Semantic Web viele neue Möglichkeiten. Da gerade in der Tourismus-Domäne das Suchen und Vernetzen von Informationen eine große Rolle spielt, kann mit dem Einsatz von semantischen Technologien die Qualität der zur Verfügung gestellten Daten optimiert werden. Das in Kapitel 3 vorgestellte niederländische Tourismus-Portal Vakantieland nutzt einige dieser Technologien. So werden sämtliche Daten zu den Point of Interests (POIs) in einer selbstentwickelten eTourismus-Ontologie gespeichert. Diese Ontologie, im speziellen das Schema für die touristischen Merkmale, war Gegenstand dieser Arbeit.

Bei der semantischen Annotation der ehemals relational vorliegenden Daten des Vakantieland-Portals wurde in erster Linie auf eine einfache und funktionale Umstellung gedacht. Mit der Zeit hat sich aber herauskristallisiert, dass vor allem die Modellierung der Merkmale nicht optimal gestaltet war, weswegen die Ontologie einer Evolution unterzogen werden sollte. In Kapitel 4 wurde daher der aktuelle Stand der Technik diskutiert. So wurde der Begriff der Ontologie-Evolution erläutert, sowie ein Prozess zur Durchführung einer Evolution. Abschließend wurden Werkzeuge zur Unterstützung des Evolutionsprozesses vorgestellt, von denen sich die Evolutionskomponente des semantischen Datenwikis OntoWiki als am besten geeignet herausstellte und somit für die Durchführung der Evolution gewählt wurde.

Aufbauend auf den Kenntnissen der bisherigen Kapitel und den Erfahrungen aus dem Einsatz der Vakantieland-Applikation wurden in Kapitel 5 zunächst einige Anwendungsszenarien vorgestellt. Auf Basis

dieser Szenarien wurden dann im Anschluss die Anforderungen für die verschiedenen zu bearbeitenden Bereiche dieser Arbeit aufgestellt. Ein Bereich betraf die Definition eines neuen Schemas für die touristischen Merkmale. Für die Umstellung der Daten vom alten zu dem neuen Schema wurden im zweiten Bereich die Anforderungen für ein Programm zur Durchführung der Evolution bestimmt. Im letzten und dritten Bereich wurden die Anforderungen für den Anwendungsfall Vakantieland spezifiziert.

Der erste Schritt zu einem neuen Schema für die Merkmale bestand in der Analyse der alten Ontologie, welche im Kapitel 6 durchgeführt wurde. Diese deckte einige Fehler auf, unter anderem Probleme in der Semantik sowie Inkonsistenzen und falsche Modellierungen. Aus diesen Erkenntnissen heraus wurden dann im gleichen Kapitel, basierend auf den ermittelten Problemfeldern, mehrere Ontologie-Entwurfsmuster für die Modellierung von touristischen Merkmalen definiert.

Im Kapitel 7 wurde zunächst eine Evolutionsstrategie bestimmt, die festlegt, zu welchem Entwurfsmuster die alten Merkmale gemäß ihrer Eigenschaften umgewandelt werden sollen. Auf dieser Grundlage wurden dann in einer Änderungsermittlung sämtliche Entscheidungen zur Evolution der Merkmale getroffen. Schließlich wurden dann Algorithmen definiert, wie die Merkmale und Instanzdaten der Ontologie in einzelnen Anpassungsschritten an das neue Schema angeglichen werden können.

Die Vorstellung der Implementierungsarbeiten in Kapitel 8 gliederte sich in drei Teile. Zunächst wurde die Evolutionskomponente von OntoWiki um eine RPC-Schnittstelle erweitert. Es wurden auch einige Optimierungen vorgenommen, da sich bei Anwendung zur Evolution herausstellte, dass der Algorithmus zur Bestimmung der zu bearbeitenden Daten unnötige und vor allem redundante Informationen ermittelte. Im zweiten Teil wurde das Programm zur Evolution der Merkmale vorgestellt und dessen Implementierung erläutert. Der dritte Teil bestand aus der Anpassung der Vakantieland-Applikation an das neue Schema sowie der Integration einer fassetierten Suche für die Merkmale. Diese ermöglicht nun, zusätzlich zu den bisherigen Filterfunktionen nach Lage und Klassifizierung, eine Einschränkung der POIs nach deren Merkmalen.

9.1 EVALUATION

In diesem Abschnitt sollen die getätigten Arbeiten hinsichtlich der definierten Anforderungen evaluiert werden. Dazu soll auch eine Bewertung stattfinden, inwieweit die in Kapitel 5 vorgestellten Anwendungsszenarien zum Abschluss dieser Arbeit erfüllt werden.

Eines der Ziele (siehe Abschnitt 1.2) bestand in einer deutlichen Reduzierung der Merkmale und einer somit einhergehenden Vereinfachung der Merkmals-Hierarchie. Im Zuge dieser Arbeit konnte nun die Zahl der Merkmale von ehemals circa 1200 auf knapp 130 reduziert werden. Ein weiteres Ziel war die einheitliche Modellierung und dadurch bessere Wartbarkeit der Merkmale und somit auch der eTourismus-Ontologie und ihrer Instanzdaten. Mit Hilfe der Ontologie-Entwurfsmuster stehen einem Administrator nun konkrete Vorgaben zur Modellierung der Merkmale zur Verfügung. Somit wird einheitliches Schema garantiert, auch wenn verschiedene Administratoren Änderungen vornehmen. Für die Artefakte, welche auf der Ontologie aufbauen, wird die Verarbeitung der Informationen ebenfalls vereinfacht. Das Auslesen und Darstellen der Merkmalsinformationen muss lediglich an die vorgegebene Struktur der Entwurfsmuster angepasst werden. Alle zukünftigen Merkmale, die nach diesen Mustern modelliert werden, können durch die auf diesem Schema aufbauenden Artefakte, im konkreten Fall dieser Arbeit das Vakantieland-Portal, ohne zusätzliche Anpassungen verarbeitet werden.

Durch die Implementierung eines Evolutionsprogramms konnte eine Evolution der Merkmale erfolgreich durchgeführt werden. Das Programm bietet durch umfangreiche Konfigurationsmöglichkeiten eine leichte Anpassung an sich eventuell ändernde Anforderungen und kann daher auch für zukünftige Evolutionsschritte verwendet werden. So könnten beispielsweise die bisher nicht einer Evolution unterzogenen oder die für eine spätere Bearbeitung markierten Merkmale mit Hilfe des Programms angepasst werden. Damit eventuell neu hinzugekommen Daten erhalten bleiben beziehungsweise die Evolution nicht von Grund auf neu durchgeführt werden muss, können zum Beispiel sämtliche Angaben zu den bereits einer Evolution unterzogenen Merkmalen aus der Konfiguration entfernt werden. Somit können zuvor ausgelassene oder auch neue Merkmale durch das Programm angepasst werden.

Das Anwendungsszenario 5.1.3 beschrieb die Planung einer Radtour mit dem Tourismus-Portal Vakantieland. In der Version vor dieser Arbeit bot Vakantieland keine Möglichkeit zur Filterung der POIs nach deren Merkmalen, wodurch das Szenario nicht erfüllt werden konnte. Durch eine Integration einer fassetierten Suche für die touristischen Merkmale in die Startseite der Applikation ist dies nun aber möglich. Ein Tourist kann nun parallel zu den Filtermöglichkeiten nach der Klassifizierung der POIs, deren Lage und der Landkarte auch nach den Merkmalen filtern, womit weitaus mächtigere Filteranfragen erstellt werden können. Dies bietet potenziellen Touristen somit einen deutlichen Mehrwert und die Begründung für den Einsatz von semantischen Technologien durch die Vakantieland-Applikation wird weiter untermauert. Die Abbildung 21 zeigt eine Filterung von POIs ähnlich den Angaben aus dem Anwendungsszenario.

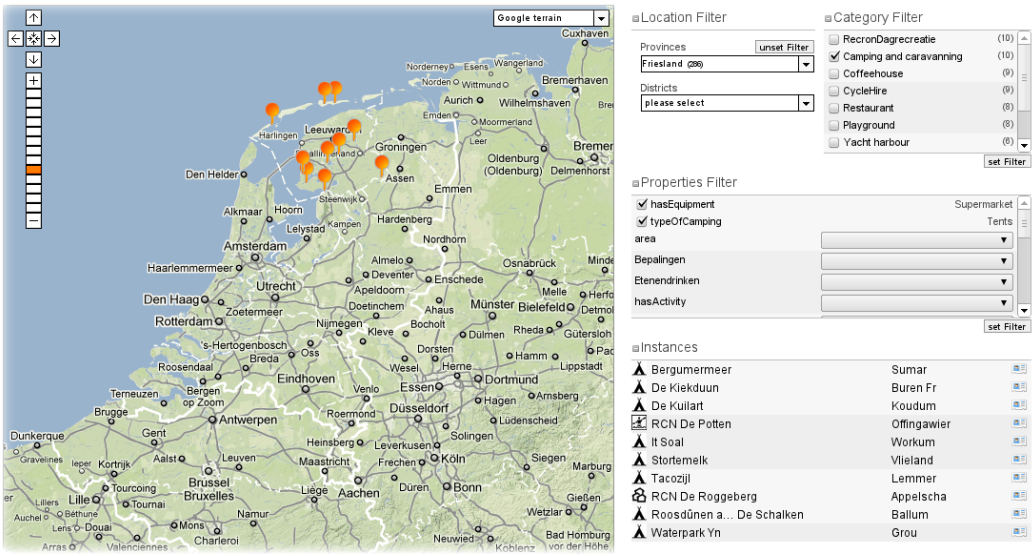


Abbildung 21: Eine Auswahl an Filterkriterien, welche denen aus dem Anwendungsszenario 5.1.3 entspricht. Insgesamt treffen mit dem aktuellen Datenbestand 10 POIs auf die Suchanfrage zu.

9.2 AUSBLICK

Bei der in dieser Arbeit durchgeführten Evolution wurden bekanntlich nicht alle Merkmale bearbeitet. So wurden einige ohne Anpassungen in das neue Schema übernommen und andere für eine spätere Bearbeitung markiert. Von daher sollten diese Merkmale in einer zukünftigen Evolution berücksichtigt werden. Dies würde auch eine weitere Überarbeitung des Schemas mit sich bringen. Falls es dazu kommt, sollte ebenfalls eine Änderung der Art, wie die Merkmale als solche deklariert werden, in Betracht gezogen werden. Bisher werden alle touristischen Merkmale in der Ontologie durch eine Subprädikats-Relation zu `etf:feature` als Merkmal deklariert. Das Prädikat `etf:feature` repräsentiert, wie auch im Bezeichner angegeben, ein Merkmal eines POI, wodurch es sich auch bei allen Subprädikaten um Merkmale handelt. Jedoch sollte entsprechend den üblichen RDF-Modellierungspraktiken `etf:feature` eher als Klasse definiert sein: „Die Klasse aller touristischen Merkmale“. Denkbar wäre also, alle Merkmale als Typ einer neuen Klasse `etc:Feature` zu deklarieren, wobei diese selbst Subklasse von `rdf:Property` sein sollte. Dadurch würde es sich automatisch bei jeder Instanz dieser Klasse um ein Prädikat handeln. Problematisch ist an dieser Stelle jedoch die Tatsache, wie für alle Merkmale der Definitionsbereich allgemein auf die Klasse `etc:POI` eingeschränkt werden kann, da der Definitionsbereich von `rdfs:domain` auf die Klasse `rdf:Property` festgelegt ist und somit nicht an Klassen verwendet werden kann. Dies ist auch der Grund dafür, warum die eben angesprochene Umstellung nicht bereits durchgeführt wurde.

Aktuell wird die eTourismus-Ontologie einer Vakantieland-Instanz durch eine dazugehörige OntoWiki-Instanz verwaltet. Mit dieser werden zum Beispiel auch neue Merkmale angelegt. Hier wäre es zur Unterstützung dieser Tätigkeit denkbar, mit Hilfe der Erweiterungsmöglichkeiten von OntoWiki eine Art Assistenten zur Erzeugung von neuen Merkmalen auf Basis der definierten Ontologie-Entwurfsmuster zu implementieren. Ein solcher Assistent würde das Anlegen neuer Merkmale erheblich vereinfachen und zudem eventuell fehlerhafte Modellierungen weiter einschränken.

LITERATURVERZEICHNIS

[Adida u. a. 2008] Adida, Ben ; Birbeck, Mark ; McCarron, Shane ; Pemberton, Steven: *RDFa in XHTML: Syntax and Processing — A Collection of Attributes and Processing Rules for Extending XHTML to Support RDF*. http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014. Version: 2008. – W3C Recommendation

[Arenas u. a. 2010] Arenas, Marcelo (Hrsg.) ; Consens, Mariano (Hrsg.) ; Mallea, Alejandro (Hrsg.): *Revisiting Blank Nodes in RDF to Avoid the Semantic Mismatch with SPARQL*. 2010 . – W3C Workshop – RDF Next Steps

[Beckett 2004] Beckett, Dave: *RDF/XML Syntax Specification (Revised)*. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Version: 2004. – W3C Recommendation

[Beckett u. Berners-Lee 2008] Beckett, Dave ; Berners-Lee, Tim: *Turtle - Terse RDF Triple Language*. http://www.w3.org/TeamSubmission/turtle/. Version: 2008. – W3C Team Submission

[Beckett u. Broekstra 2008] Beckett, Dave ; Broekstra, Jeen: *SPARQL Query Results XML Format*. http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115. Version: 2008. – W3C Recommendation

[Berners-Lee u. a. 2005] Berners-Lee, T. ; Fielding, R. ; Masinter, L.: *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. 2005

[Berners-Lee 2000] Berners-Lee, Tim: *Semantic Web - XML2000*. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html. Version: 2000

[Berners-Lee 2006] Berners-Lee, Tim: Linked Data. In: *W3C Design Issues* (2006). http://www.w3.org/DesignIssues/LinkedData.html

[Berners-Lee u. Connolly 2008] Berners-Lee, Tim ; Connolly, Dan: Notation3 (N3): A readable RDF syntax / W3C. Version: 2008. http://www.w3.org/TeamSubmission/n3/. 2008. (W3C Team Submission). – Forschungsbericht

[Berners-Lee u. Fischetti 1999] Berners-Lee, Tim ; Fischetti, Mark: *Weaving the web: The original design and ultimate destiny of the world wide web by its inventor*. Harper, 1999. – xi, 226 S.

[Berners-Lee u. a. 2001] Berners-Lee, Tim ; Hendler, James ; Lassila, Ora: The Semantic Web. In: *Scientific American* 284 (2001), Nr. 5, 34–43. http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[Bray u. a. 2008] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/PER-xml-20080205. Version: 2008. – W3C Recommendation

[Brickley u. Guha 2004] Brickley, Dan ; Guha, R.V.: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Version: 2004. – W3C Recommendation

[Clark u. a. 2008] Clark, Kendall G. ; Feigenbaum, Lee ; Torres, Elias: *SPARQL Protocol for RDF*. http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115. Version: 2008. – W3C Recommendation

[DCMI 2008] DCMI: *DCMI Metadata Terms*. http://dublincore.org/documents/2008/01/14/dcmi-terms/. Version: 2008

[English u. a. 2002] English, Jennifer ; Hearst, Marti ; Sinha, Rashmi ; Swearingen, Kirsten ; Yee, Ka-Ping: Flexible Search and Navigation Using Faceted Metadata / University of Berkeley. 2002. – Forschungsbericht

[Eyal Oren and Renaud Delbru and Stefan Decker 2006] Eyal Oren and Renaud Delbru and Stefan Decker: Extending faceted navigation for RDF data. In: *5th International Semantic Web Conference*, 2006

[Fielding u. a. 1999] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)*. 1999

[Franks u. a. 1999] Franks, J. ; Hallam-Baker, P. ; Hostetler, J. ; Lawrence, S. ; Leach, P. ; Luotonen, A. ; Stewart, L.: *HTTP Authentication: Basic and Digest Access Authentication*. 1999

[Gennari u. a. 2003] Gennari, John H. ; Musen, Mark A. ; Fergerson, Ray W. ; Grosso, William E. ; Crubézy, Monica ; Eriksson, Henrik ; Noy, Natalya F. ; Tu, Samson W.: The evolution of Protégé: an environment for knowledge-based systems development. In: *Int. J. Hum.-Comput. Stud.* 58 (2003), Nr. 1, S. 89–123

[Grant u. Beckett 2004] Grant, Jan ; Beckett, Dave: *RDF Test Cases*. http://www.w3.org/TR/2004/REC-rdf-testcases-20040210. Version: 2004. – W3C Recommendation

[Gruber 1993] Gruber, Thomas R.: A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition* 5 (1993), Nr. 2, S. 199–220

[Hayes u. McBride 2004] Hayes, Patrick ; McBride, Brian: *RDF Semantics*. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Version: 2004. – W3C Recommendation

[Klein u. Noy 2003] Klein, M. ; Noy, N.: A Component-based Framework for Ontology Evolution. In: *Proc. Workshop on Ontologies and Distributed Systems, IJCAI*, 2003

[Klyne u. Carroll 2004] Klyne, Graham ; Carroll, Jeremy J.: *Resource Description Framework (RDF): Concepts and Abstract Syntax*. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Version: 2004. – W3C Recommendation

[Manola u. Miller 2004] Manola, Frank ; Miller, Eric: *RDF Primer*. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. Version: 2004. – W3C Recommendation

[Martin 2007] Martin, Michael: Exploring the Netherlands on a Semantic Path. In: Auer, Sören (Hrsg.) ; Bizer, Christian (Hrsg.) ; Müller, Claudia (Hrsg.) ; Zhdanova, Anna (Hrsg.): *Proceedings of the 1st Conference on Social Semantic Web* Bd. P-113, Bonner Köllen Verlag, 2007 (GI-Edition - Lecture Notes in Informatics (LNI), ISSN 1617-5468)

[Martin u. Auer 2010] Martin, Michael ; Auer, Sören: Categorisation of Semantic Web Applications. In: *Proceedings of the 4th International Conference on Advances in Semantic Processing (SEMAPRO2010) 25 October – 30 October, Florence, Italy*, 2010

[Martin u. a. 2010] Martin, Michael ; Unbehauen, Jörg ; Auer, Sören: Improving the Performance of Semantic Web Applications with SPARQL Query Caching. In: *Proceedings of 7th Extended Semantic Web Conference (ESWC 2010), 30 May – 3 June 2010, Heraklion, Greece*, 2010

[Miles 2007] Miles, Alistair: *Patterns for Working With SKOS and OWL*. http://isegserv.itd.rl.ac.uk/public/skos/2007/10/f2f/skos-owl-patterns.html. Version: 2007. – SWDWG Amsterdam F2F October 2007

[Miles u. Bechhofer 2008] Miles, Alistair ; Bechhofer, Sean: *SKOS Simple Knowledge Organization System Reference*. http://www.w3.org/TR/2009/REC-skos-reference-20090818/. Version: 2008. – W3C Recommendation

[Motik u. a. 2008] Motik, Boris ; Patel-Schneider, Peter F. ; Parsia, Bijan: *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. http://www.w3.org/TR/2008/WD-owl2-syntax-20081202. Version: 2008. – W3C Recommendation

[Noy u. a. 2004] Noy, N. F. ; Kunnatur, S. ; Klein, M. ; Musen, M.A.: Tracking Changes During Ontology Evolution. In: *Third International Conference on the Semantic Web (ISWC)*, 2004

[Noy u. Rector 2006] Noy, Natasha ; Rector, Alan: *Defining N-ary Relations on the Semantic Web*. http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/. Version: 2006. – W3C Working Group Note

[Prud'hommeaux u. Seaborne 2008] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF*. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Version: 2008. – W3C Recommendation

[Rieß 2010] Riess, Christoph: *Evolution und Refactoring von RDF-Graphen basierend auf Pattern*, Universität Leipzig, Fakultät für Mathematik und Informatik, Institut für Informatik, Diplomarbeit, 2010

[Rieß u. a. 2010] Riess, Christoph ; Heino, Norman ; Tramp, Sebastian ; Auer, Sören: EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In: *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Springer, 2010 (Lecture Notes in Computer Science)

[Seaborne u. a. 2008] Seaborne, Andy ; Manjunath, Geetha ; Bizer, Chris ; Breslin, John ; Das, Souripriya ; Davis, Ian ; Harris, Steve ; Idehen, Kingsley ; Corby, Olivier ; Kjernsmo, Kjetil ; Nowack, Benjamin: SPARQL/Update A language for updating RDF graphs. Version: 2008. http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/. 2008. – Forschungsbericht. – W3C Member Submission

[Stojanovic u. Motik 2002] Stojanovic, L. ; Motik, B.: Ontology Evolution within Ontology Editors. In: *EKAW'02/EON Workshop*, 2002, S. 53–62

[Stojanovic u. a. 2002] Stojanovic, Ljiljana ; Maedche, Alexander ; Motik, Boris ; Stojanovic, Nenad: User-Driven Ontology Evolution Management. In: *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management.*, Springer-Verlag, 2002, S. 285–300

[Stojanovic u. Stojanovic 2002] Stojanovic, Nenad ; Stojanovic, Ljiljana: An Approach for the Evolution of Ontology-based Knowledge Management Systems. In: *EKAW 2002 Workshop on Knowledge Management*

through Corporate Semantic Webs, at 13th European Conference on Knowledge Engineering and Knowledge Management EKAW Madrid, Spain, 2002, S. 22–37

[Tramp u. a. 2010a] Tramp, Sebastian ; Frischmuth, Philipp ; Heino, Norman: OntoWiki – a Semantic Data Wiki Enabling the Collaborative Creation and (Linked Data) Publication of RDF Knowledge Bases. In: *Demo Proceedings of the EKAW 2010*, 2010

[Tramp u. a. 2010b] Tramp, Sebastian ; Heino, Norman ; Auer, Sören ; Frischmuth, Philipp: Making the Semantic Data Web easily writeable with RDFauthor. In: *Proceedings of 7th Extended Semantic Web Conference (ESWC 2010), 30 May – 3 June 2010, Heraklion, Greece* Bd. 6089 Springer, 2010 (Lecture Notes in Computer Science), 436–440

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGVERZEICHNIS

A ANHANG

A.1 ENTSCHEIDUNGEN AUS DER ÄNDERUNGSERMITTLUNG

Die folgende Tabelle gibt einen Überblick über sämtliche Entscheidungen zur Evolution der Merkmale, die während der Änderungsermittlung getroffen wurden. In der ersten Spalte sind sowohl die Domänenmerkmale (Knoten in der alten Hierarchie), welche mehrere Submerkmale zu einer Domäne zusammenfassen, als auch einzelne Merkmale (Blätter innerhalb der alten Hierarchie), die gesondert betrachtet werden müssen, angegeben. Das Ontologie-Entwurfsmuser, welches jeweils angewandt werden soll, ist in der zweiten Spalte notiert. In der dritten Spalte stehen Außnahmen oder Hinweise. Hier wird beispielsweise angegeben, ob bei der Evolution dieser Gruppe von Merkmalen (Domäne) etwas beachtet werden muss. Aus Platzgründen werden die einzelnen Ontologie-Entwurfsmuster in abgekürzter Form genannt. *EW* steht dabei für das Entwurfsmuster „Merkmale mit Einfachen Werten“, *KW* für „Merkmale mit Komplexen Werten“ und *Poly* für „Polyvalente Merkmale“.

Merkmal/Domäne	Entwurfsmuster	Hinweise
etf:feature	—	keine Änderung
etf:Aangeslotenbij	KW, konkret	—
etf:Accommodation-recreation	—	löschen
etf:Adverteren	—	überarbeiten
etf:Agrarischbedrijf	KW, abstrakt	—
etf:Algemeensanitair	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Algemene-voorzieningen	KW, abstrakt	etf:Halls.number → EW
etf:Ambiance	KW, abstrakt	—
etf:Asian.Asiatic	—	löschen
etf:Bedrijfstype	—	überarbeiten
etf:Beds	EW	—
etf:Bepalingen	—	unverändert
etf:Bereikbaarheid	KW, abstrakt	zusammenfügen mit etf:Ligginginbij

Fortsetzung auf nächster Seite

Fortgesetzt von vorheriger Seite

Merkmal/Domäne	Entwurfsmuster	Hinweise
etf:Beschutting	KW, abstrakt	—
etf:Betaalmiddelen	KW, konkret	—
etf:Bijzonderegebouwen	KW, abstrakt	etf:Intheyear → EW
etf:Bodemgesteldheid	KW, abstrakt	—
etf:Buitenlandsevaluta	KW, konkret	—
etf:Bungalow.personen	KW, abstrakt	—
etf:Camping.plaatsen	KW, abstrakt	—
etf:Camping.type	KW, abstrakt	—
etf:Camponafarm	—	löschen
etf:Campsite	—	löschen, etf:Ampere mit etf:Electricity verschmelzen, etf:Electricity→ EW
etf:Classificatie	KW, konkret	etf:#Flag(s) mit etf:#Star(s) verschmelzen
etf:Companyopen-tothepublic	KW, abstrakt	—
etf:Congresscentre	—	löschen
etf:Dagrecreatie.weer	—	überarbeiten
etf:Daytrip.s.daysouting-.s.one-dayouting.s	—	löschen
etf:Dienstverlening	KW, abstrakt	—
etf:Doelgroep	KW, abstrakt	—
etf:Electricity	EW	—
etf:Erkenningen	Poly	—
etf:Erkenningen.camping	Poly	zusammenfügen mit etf:Erkenningen
etf:Erkenningen.hotel	Poly	zusammenfügen mit etf:Erkenningen
etf:Erkenningen.-restaurant	Poly	zusammenfügen mit etf:Erkenningen
etf:Etenendrinken	—	unverändert
etf:European	—	löschen
etf:Fine	—	löschen
etf:Groepsaccomodatie	—	unverändert
etf:Groupfacility	—	löschen
etf:Halls.number	EW	—
etf:Hemelrijk	—	löschen
etf:Holidaypark	—	unverändert
etf:Horeca	KW, abstrakt	—
etf:Hotel	—	löschen, etf:Rooms → EW, etf:Beds → EW

Fortsetzung auf nächster Seite

Fortgesetzt von vorheriger Seite

Merkmal/Domäne	Entwurfsmuster	Hinweise
etf:Hotel.type	KW, abstrakt	—
etf:Huisdieren	—	unverändert
etf:Inrichting	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Intheyear	EW	—
etf:Keten	KW, konkret	—
etf:Ligginginbij	KW, abstrakt	—
etf:Logiesenontbijt	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Milieu.barometer	—	überarbeiten
etf:Musea.categorie	KW, abstrakt	—
etf:Musea.collectie	KW, abstrakt	—
etf:Museum	—	löschen
etf:Ontspanning	KW, abstrakt	—
etf:Recron	—	löschen
etf:Reserverenvia	KW, konkret	—
etf:Rest.keuken	KW, abstrakt	—
etf:Rest.keuken.europees	KW, abstrakt	zusammenfügen mit etf:Rest.keuken
etf:Rest.keuken.ziatisch	KW, abstrakt	zusammenfügen mit etf:Rest.keuken
etf:Rest.soort	KW, abstrakt	—
etf:Restaurant	—	unverändert
etf:Rooms (Domäne)	—	unverändert
etf:Rooms	EW	umbennen zwecks Doppeldeutigkeit
etf:root	—	löschen
etf:Rust	—	überarbeiten
etf:Speciaalterrein	—	unverändert
etf:Speciaalvoorkinderen	KW, abstrakt	—
etf:Sport	KW, abstrakt	zusammenfügen mit etf:Ontspanning
etf:Startpagina	—	überarbeiten
etf:Statistieken	—	löschen
etf:Tarief	—	überarbeiten
etf:Tarief.bungalows	—	überarbeiten
etf:Tarief.campings	—	überarbeiten
etf:Tarief.dagrecreatie	—	überarbeiten
etf:Tarief.groeps-accomodaties	—	überarbeiten

Fortsetzung auf nächster Seite

Fortgesetzt von vorheriger Seite

Merkmal/Domäne	Entwurfsmuster	Hinweise
etf:Tarief.hotels	—	überarbeiten
etf:Tarief.jachthavens	—	überarbeiten
etf:Tarief.restaurants	—	überarbeiten
etf:Terreininrichting	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Verhuur	KW, abstrakt	—
etf:Verzorging	KW, abstrakt	—
etf:Voorzieningen-gehandicapten	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Voorzieningen-perkamer	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Wasvoorzieningen	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen
etf:Watersport	KW, abstrakt	zusammenfügen mit etf:Ontspanning
etf:Woning.constructie	KW, abstrakt	zusammenfügen mit etf:Bungalow.personen
etf:Yacht-basin%2Cmarina	KW, abstrakt	—
etf:Yacht-basin.marina	—	löschen
etf:Zalencentra	KW, abstrakt	—
etf:Zwemgelegenheid	KW, abstrakt	zusammenfügen mit etf:Algemenevoorzieningen

Tabelle 10: Tabelle mit allen Entscheidungen und Hinweisen, wie welche Merkmalsdomänen und Einzelmerkmale durch eine Evolution an das neue Schema der Merkmale angepasst werden sollen.

A.2 DEFINIERTE EVOLUTIONSMUSTER

Die Tabelle 11 enthält alle für die Evolution der Merkmale benötigten Evolutionsmuster mit einer kurzen Beschreibung.

A.3 QUELLCODE DER EVOLUTIONSKOMPONENTE VON ONTOWIKI

Der Quellcode der Evolutionskomponente wird im Google Code-Projekt von OntoWiki gepflegt, welches unter der Adresse

`http://code.google.com/p/ontowiki/`

erreichbar ist. Die in dieser Arbeit entwickelten Erweiterungen wurden direkt in den eigens für die Evolutionskomponente erstellten Entwick-

lungszweig „Feature-EvolutionEngine“ eingepflegt. Eine lokale Kopie einer funktionstüchtigen Version von OntoWiki mit der Evolutionskomponente kann über den Befehl

```
1 $ hg clone http://ontowiki.googlecode.com/hg/ ontowiki -r
    d295538cc0
```

erstellt werden. Die zuletzt erfolgreich eingesetzte Version entspricht der Revision `d295538cc0`, welche auch alle Änderungen aus dieser Arbeit enthält.

A.4 QUELLCODE DER VAKANTIELAND-APPLIKATION

Der Quellcode für die in dieser Arbeit angepassten Vakantieland-Applikation wird derzeit in einem eigenständigen Entwicklungszweig „`branches/featureEvolution`“ gepflegt. Eine lokale Kopie kann über den Befehl

```
1 $ svn co
    http://svn.aksw.org/vakantieland/branches/featureEvolution
    -r 1423
```

erstellt werden. Der Zustand zur Abgabe dieser Arbeit entspricht der Revision `1423` des entsprechenden Entwicklungszweigs.

Muster	Beschreibung
adaptPoisTo-ComplexValue.json	Passt die Merkmalsangaben der POIs auf Instanz-Ebene entsprechend dem Muster „Merkmale mit Komplexen Werten“ an.
adaptPoisToPolyvalent-ComplexValue.json	Passt die Merkmalsangaben der POIs auf Instanz-Ebene entsprechend dem Muster „Polyvalente Merkmale“ an.
adaptPoisTo-SimpleValue.json	Passt die Merkmalsangaben der POIs auf Instanz-Ebene entsprechend dem Muster „Merkmale mit Einfachen Werten“ an.
copyDcDescriptions.json	Kopiert alle Beschreibungen, angegeben durch `dc:description`, von einer Ressource zu einer anderen.
copyFeatureIcon.json	Kopiert alle Icons, angegeben durch `vakp:hasIcon`, von einer Ressource zu einer anderen.
copyRdfsLabels.json	Kopiert alle durch `rdfs:label` angegebenen Bezeichner von einer zu einer anderen Ressource.
copyRdfsComments.json	Kopiert alle durch `rdfs:comment` angegebenen Kommentare von einer zu einer anderen Ressource.
createFeature.json	Erzeugt ein neues Merkmal.
createOwlClass.json	Erzeugt eine neue Ressource vom Typ `owl:Class`
createSubClass.json	Stellt eine `rdfs:subClassOf`-Relation zwischen zwei Klassen her.
deleteObject.json	Löscht alle Tripel, in denen die angegebene Ressource als Objekt auftritt.
deleteProperty.json	Löscht alle Tripel, in denen die angegebene Ressource als Prädikat auftritt.
deleteSubfeatures.json	Löscht alle Submerkmale und deren Verwendungen auf Instanz-Ebene an den POIs des angegebenen Merkmals.
deleteSubject.json	Löscht alle Tripel, in denen die angegebene Ressource als Subjekt auftritt.
mergeFeatures.json	Führt zwei Merkmale zusammen, in dem alle Verwendungen auf Instanz-Ebene des einen angegebenen Merkmals durch das zweite angegebene ersetzt werden.
refactorFeature.json	Markiert das angegebene Merkmal für eine spätere Bearbeitung.
refactorSubfeatures.json	Markiert alle Submerkmale des angegebenen Merkmals für eine spätere Bearbeitung.
transformFeatures-ToTourismObjects.json	Wandelt alle Submerkmale des angegebenen Merkmals zu Komplexen Werten um.

Tabelle 11: Übersicht der definierten Evolutionsmuster.

ERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, 8. November 2010

Marvin Frommhold